MIL-STD-1777
12 AUGUST 1983

MILITARY STANDARD

INTERNET PROTOCOL

NO DELIVERABLE DATA REQUIRED BY THIS DOCUMENT

DEPARTMENT OF DEFENSE
WASHINGTON, D.C. 20301

Internet Protocol

MIL-STD-1777

1. This Military Standard is approved for use by all Departments and Agencies of the Department of Defense.

2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Defense Communications Agency, ATTN: J110, 1860 Wiehle Avenue, Reston, Virginia 22090, by using the self-addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document, or by letter.

FOREWORD

This document specifies the Internet Protocol (IP) which supports the interconnection of communication subnetworks. The document includes an introduction to IP with a model of operation, a definition of services provided to users, and a description of the architectural and environmental requirements. The protocol services interfaces and mechanisms are specified using an abstract state machine model.

CONTENTS

CONTENTS

CONTENTS - Continued

CONTENTS - Continued

CONTENTS - Continued

1. SCOPE

1.1 Purpose. This standard establishes criteria for the Internet Protocol (IP) which supports the interconnection of communication subnetworks.

1.2 Organization. This standard introduces the Internet Protocol's role and purpose, defines the services provided to users, and specifies the mechanisms needed to support those services. This standard also defines the services required of the lower protocol layer, describes the upper and lower interfaces, and outlines the execution environment services needed for implementation.

1.3 Application. The Internet Protocol (IP) and the Transmission Control Protocol (TCP) are mandatory for use in all DoD packet switching networks which connect or have the potential for utilizing connectivity across network or subnetwork boundaries. Network elements (hosts, front-ends, bus interface units, gateways, etc.) within such networks which are to be used for inter-netting shall implement TCP/IP. The term network as used herein includes Local Area Networks (LANs)) but not integrated weapons systems. Use of TCP/IP within LANs is strongly encouraged particularly where a need is perceived for equipment interchangeability or network survivability. Use of TCP/IP in weapons systems is also encouraged where such usage does not diminish network performance.

2. REFERENCED DOCUMENTS

2.1 Issues of documents. The following documents of the issue in effect on date of invitation for bids or request for proposal, form a part of this standard to the extent specified herein. (The provisions of this paragraph are under consideration.)

2.2 Other publications. The following documents form a part of this standard to the extent specified herein. Unless otherwise indicated, the issue in effect on date of invitation for bids or request for proposals shall apply. (The provisions of this paragraph are under consideration.)

3. DEFINITIONS

3.1 Definition of terms. The definition of terms used in this standard shall comply with FED-STD-1037. Terms and definitions unique to MIL-STD-1777 are contained herein.

a. Datagram. A self-contained package of data carrying enough information to be routed from source to destination without reliance on earlier exchanges between source or destination and the transporting subnetwork.

b. Datagram fragment. The result of fragmenting a datagram, also simply referred to as a fragment. A datagram fragment carries a portion of data from the larger original, and a copy of the origi-nal datagram header. The header fragmentation fields are adjusted to indicate the fragments relative position within the original datagram.

c. Datagram service. A datagram, defined above, delivered in such a way that the receiver can determine the boundaries of the datagram as it was entered by the source. A datagram is delivered with non-zero probability to the desired destination. The sequence in which datagrams are entered into the subnetwork by a source is not necessarily preserved upon delivery at the destination.

d. Destination. An IP header field containing an internet address indicating where a datagram is to be sent.

e. DF. Don't Fragment flag: An IP header field that when set “true” prohibits an IP module from fragmenting a datagram to accomplish delivery.

f. EFTP. Electronic File Transfer Protocol. Electronic mail.

g. Fragmentation. The process of breaking the data within a data- gram into smaller pieces and attaching new internet headers to form smaller datagrams.

h. Fragment Offset. A field in the IP header marking the relative position of a datagram fragment within the larger original datagram.

i. FTP. File Transfer Protocol.

j. Gateway. A device, or pair of devices, which interconnect two or more subnetworks enabling the passage of data from one subnetwork to another. In this architecture, a gateway usually contains an IP module, a Gateway-to-Gateway Protocol (GGP) module, and a subnetwork protocol module (SNP) for each connected subnetwork.

k. Header. Collection of control Information transmitted with data between poor entities.

l. Host. A computer which is a source or destination of messages from the point of view of the communication subnetwork.

a. ICMP. Internet Control Message Protocol, the collection of error conditions and error message formats exchanged by IP modules in both hosts and gateways.

n. Identification. An IP header field used in reassembling fragments of a datagram.

o. IHL. Internet Header Length: an IP header field indicating the number of 32-bit words making up the Internet header.

p. Internet address. A four octet (32 bit) source or destination address composed of a Network field and a REST field. The latter usually contains a local subnetwork address.

q. Internet datagram. The package exchanged between a pair of IP modules. It is made up of an IP header and a data portion.

r. Local address. The address of a host within a subnetwork. The actual mapping of an Internet address onto local subnetwork addresses is quite general. allowing for many to one mappings.

a. Local subnetwork. The subnetwork directly attached to host or gateway.

t. MF. More Fragments flag: an IP header field indicating whether a datagram fragment contains the end of a datagram.

u. MTU. Maximum Transmission Unit: a subnetwork dependent value which indicates the largest datagram that a subnetwork can handle.

v. Octet. An eight-bit byte.

v. Options. The optional set of fields at the end of the IP header used to carry control or routing data. An Options field may contain none, one, or several options, and each option may be one to several octets in length. The options allow ULPs to customize IP’s services. The options are also useful in testing situations to carry diagnostic data such as timestamps.

x. Packet network. A network based on packet-switching technology. Messages are split into small units (packets) to be routed independently on a store and forward basis. This approach pipelines packet transmission to effectively use circuit bandwidth.

y. Padding. An IP header field, one octet in length. Inserted after the last option field to ensure that the data portion of a datagram begins on a 32-bit word boundary. The Padding field value is zero.

z. Protocol. An internet header field used to identify the upper layer-protocol that is the source and destination of the data within an IP datagram.

aa. Reassembly. The process of piecing together datagram fragments to reproduce the original large datagram. Reassembly is based on fragmentation data carried in their IP headers.

bb. Reliability. One of the service quality parameters provided by the type of service mechanism. The reliability parameter can be set to one of four levels: lowest, lower, higher, or highest. It appears as a two-bit field within the Type of Service field in the IP header.

cc. Rest. The three-octet field of the internet address usually containing a local address.

dd. Segment. The unit of data exchanged by TCP modules. This term may also be used to describe the unit of exchange between any transport protocol modules.

ee. Source. An IP header field containing the Internet address of the datagram's point of origin.

ff. Stream delivery service. The special handling required for a class of volatile periodic traffic typified by voice. The class requires the maximum acceptable delay to be only slightly larger than the minimum propagation time, or requires the allowable variance in packet interarrival time to be small.

gg. SNP. Subnetwork Protocol: the protocol residing in the subnetwork layer below IP which provides data transfer through the local subnet. In some systems, an adapter module must be inserted between IP and the subnetwork protocol to reconcile their dissimilar interfaces.

hh. TCP. Transmission Control Protocol: a transport protocol providing connection-oriented, end-to-end reliable data transmission in packet-switched computer subnetworks and internetworks.

ii. TCP segment. The unit of data exchanged between TCP modules (including the TCP header).

jj. Total Length. An IP header field contain number of octets in an internet datagram, including both the IP header and the data portion.

kk. Type of Service. An IP header field containing the transmission quality parameters: precedence level, reliability level, speed level, resource trade-off (precedence vs. reliability), and transmission mode (datagram vs. stream). This field is used by the type of service mechanism which allows ULPs to select the quality of transmission for a datagram through the internet.

ll. UDP. User Datagram Protocol.

mm. ULP. Upper Layer Protocol: any protocol above IP in the layered protocol hierarchy that uses IP. This term includes transport layer protocols, presentation layer protocols, session layer protocols, and application programs.

nn. Version. An IP header field indicating the format of the EP header.

4. GENERAL REQUIREMENTS

4.1 Design. The Internet Protocol is designed to interconnect packet-switched communication subnetworks to form an internetwork. The IP transmits blocks of data, called Internet datagrams, from sources to destinations throughout the Internet. Sources and destinations are hosts located on either the same subnetwork or connected subnetworks. The IP is purposely limited in scope to provide the basic functions necessary to deliver a block of data. Each Internet datagram is an independent entity unrelated to any other Internet datagram. The IP does not create connections or logical circuits and has no mechanisms to promote data reliability, flow control, sequencing, or other services commonly found in virtual circuit protocols.

4.2 Internet protocol definition. This standard specifies a host IP. As defined in the DoD architectural model, the Internet Protocol resides in the internetwork layer. Thus, the IP provides services to transport layer protocols and relies on the services of the lower network layer protocol (See figure 1). In each gateway (a system interconnecting two or more subnets) an IP resides above two or more subnetwork protocol entities. Gateways implement Internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) to coordinate routing and other internet control information.

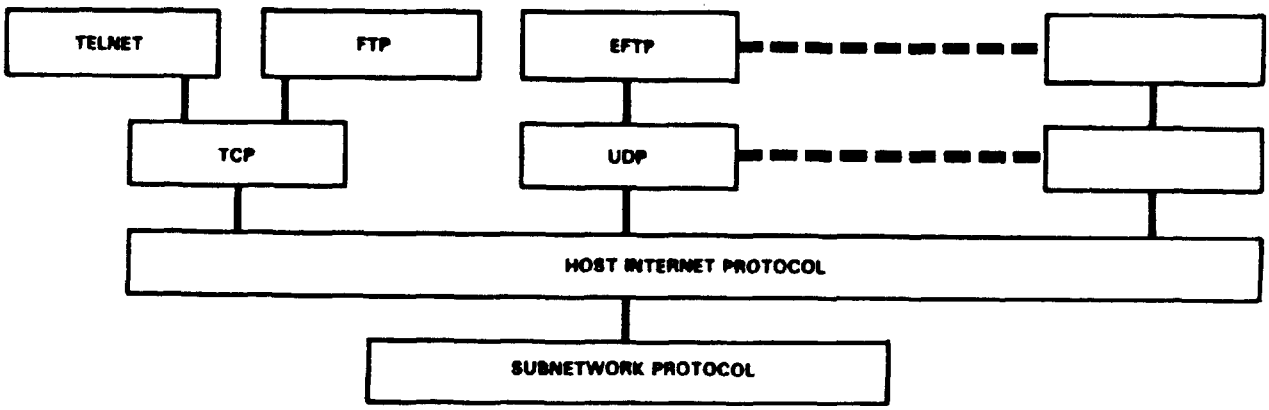


FIGURE 1. Example host protocol hierarchy.

4.2.1 Protocol implementation. In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module (See figure 2).

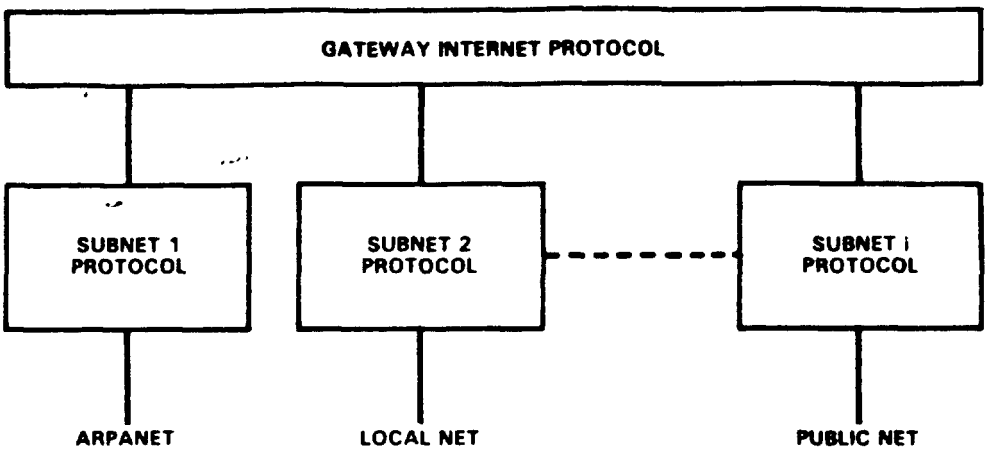


FIGURE 2. Example gateway protocol hierarchy.

4.2.2 Upper layer protocol. A protocol in an upper layer passes data to IP for delivery. IP packages the data as an internet datagram and passes it to the local subnetwork protocol for transmission across the local subnet If the destination host is on the local subnet, IP sends the datagram through the subnet directly to that host. If the destination host is on a foreign subnet, IP sends the datagram to a local gateway. The gateway, in turn, sends the datagram through the next subnet to the destination host, or to another gateway. Thus, datagrams move from one IP module to another through an interconnected set of subnetworks until they reach their destinations. The sequence of IP modules handling the datagram in transit is called the gateway route. The gateway route is distinct from the lower level node-to-node route supplied by a particular subnetwork. The gateway route is based on the destination internet address. The IP modules share common rules for interpreting internet addresses to perform internet routing.

4.2.3 Datagram processing error. Occasionally, a gateway IP or destination IP will encounter an error during datagram processing. Errors detected may be reported via the Internet Control Message Protocol (ICMP) which is implemented in the internet protocol module.

4.2.4 Fragmentation and reassembly mechanisms. In transit, datagrams may traverse a subnetwork whose maximum packet size is smaller than the size of the datagram. To handle this condition, IP provides fragmentation and reassembly mechanisms. The gateway at the smaller-packet subnet fragments the original datagram into pieces, called datagram fragments, that are small enough for transmission. The IP module in the destination host reassembles the datagram fragments to reproduce the original datagram. IP can support a diverse set of upper layer protocols (ULPs). A transport protocol with real-time requirements, such as the Network Voice Protocol (NVP), can make use of IP's datagram service directly. A transport protocol providing ordered reliable delivery, such as TCP, can build additional mechanisms on top of IP's basic datagram service. Also, IP's delivery service can be customized in some ways to suit the special needs of an upper layer protocol. For example, a predefined gateway route, called a source route, can be supplied for an individual datagram. Each IP module forwards the datagram according to the source route in addition to using the standard routing mechanism.

4.2.5 IP evolution. The current Internet Protocol evolved from proposals within the International Federation for Information Processing (IFIP) Technical Committee 6.1, in which internet functions and reliable transport functions were combined in a single protocol. Subsequent development of other ULPs (such as packet speech) led to the separation of these functions to form IP and the Transmission Control Protocol.

4.3 Scenario. The following scenario illustrates the model of operation for transmitting a datagram from one upper layer protocol to another. The scenario is purposely simple so that IP's basic operation is not obscured by the details of interface parameters or header fields.

4.3.1 Basic model of operation. A ULP in host A is to send data to its peer protocol in host B on another subnetwork. In this case, the source and destination hosts are on subnetworks directly connected by a gateway.

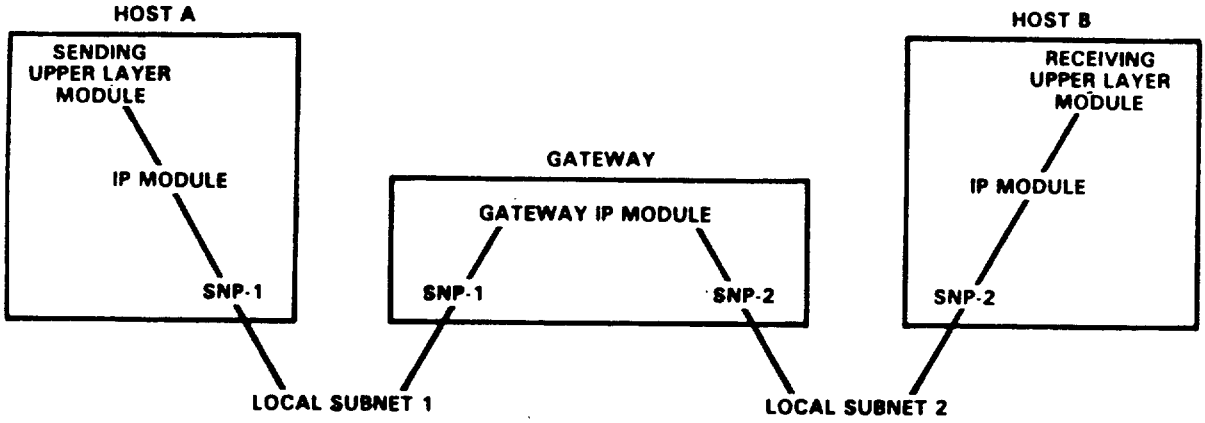


FIGURE 3. Basic model of operation.

a. The sending ULP passes its data to the IP module, along with the destination internet address and other parameters.

b. The IP module prepares an IP header and attaches the ULP's data to form an internet datagram. Then, the IP module determines a local subnetwork address from the destination internet address. In this case, it is the address of the gateway to the destination subnetwork. The internet datagram along with the local subnet address is passed to the local subnetwork protocol (SNP).

c. The SNP creates a local subnetwork header and attaches it to the datagram forming a subnetwork packet. The SNP then transmits the packet across the local subnet.

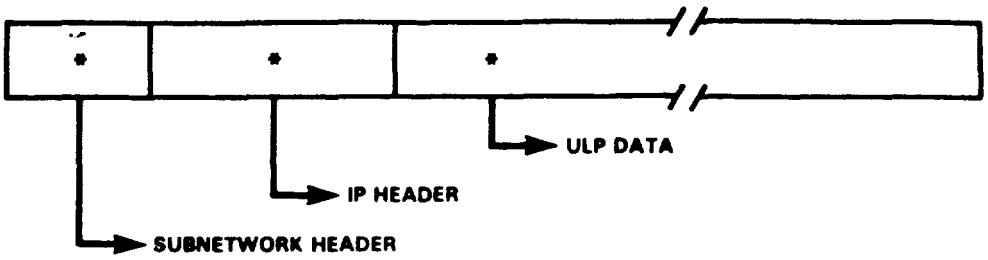


FIGURE 4. Subnetwork packet.

d. The packet arrives at the gateway connecting the first and second subnetworks. The SNP of the first subnet strips off the local subnetwork header and passes the remainder to the IP module.

a. The IP module determines from the destination internet address in the IP header that the datagram Is intended for a host in the second subnet. The IP module than derives a local subnetwork address for the destination host. That address is passed along with the datagram to the SUP of the second subnetwork for delivery.

f. The second subnet's SNP builds a local subnetwork header and appends the datagram to form a packet for the second subnetwork. That packet is transmitted across the second subnet to the destination host.

g. The SNP of the destination host strips off the local subnetwork header and hands the remaining datagram to the IP module.

h. The IP nodule determines that the datagram is bound for a ULP within this host. The data portion of the datagram and information from the IP header are passed to the ULP.

Delivery of data across the internet is complete.

5. SERVICES PROVIDED TO UPPER LAYER

5.1 Description. This section describes the services offered by the Internet Protocol to upper layer protocols (ULPs). The goals of this section are to provide the motivation for protocol mechanisms and a definition of the functions provided by this protocol. The services provided by IP are: internet datagram service, virtual network service, and error reporting service. A description of each service follows:

5.2 Datagram service. The Internet Protocol shall provide a datagram service between homogeneous upper layer protocols in an Internet environment. A datagram service is characterized by data delivery to the destination with non-zero probability; some data may possibly be lost or duplicated. Also, a datagram service does not necessarily preserve the sequence in which data is supplied by the source upon delivery at the destination.

5.2.1 Delivery service. IP shall deliver received data to a destination ULP in the same form as sent by the source ULP. IP shall discard datagrams when insufficient resources are available for processing. IP does not detect datagrams lost or discarded by the subnetwork layer. As part of the delivery service, IP insulates upper layer protocols from subnetwork-specific characteristics. For example, IP saps Internet addresses supplied by ULPs into local addresses used by the local subnetwork. Also, IP hides any packet-size restrictions of subnetworks along the transmission path within the Internet.

5.3 Generalized network services. IP shall provide to upper layer protocols the ability to select virtual network service parameters. IP shall provide a general command set for the ULPs to indicate the services desired. Thus, the ULPs can tune certain properties of IP and the underlying subnetworks to customize the transmission service according to their needs.

5.3.1 Network parameters. The virtual network parameters fall into two categories: service quality parameters and service options. Service quality parameters influence the transmission service provided by the subnetworks; service options are additional functions provided by IP. A brief description of each follows:

- Service Quality Parameters
    - Precedence: attempts preferential treatment for high importance datagrams
    - Transmission Mode: datagram vs. stream. Stream mode attempts to minimize delay and delay dispersion through reservation of network resources
    - Reliability: attempts to minimize data lose and error rate
    - Speed: attempts prompt delivery

        - Resource Tradeoff: indicates relative importance of speed vs. reliability

    - Service Options

        - Security Labeling: identifies datagram for compartmented hosts

        - Source Routing: selects set of gateway IP modules to visit in transit

        - Route Recording: records gateway IP modules encountered in transit

        - Stream Identification: names reserved resources used for stream service

        - Timestamping: records time information

        - Don’t Fragment: marks a datagram as an indivisible unit

5.4 Error reporting service. IP shall provide error reports to the upper layer protocols indicating errors detected in providing the above services. In addition, certain errors detected by lower layer protocols or supplied in ICMP messages shall be passed to the ULPs. These reports indicate several classes of errors including invalid arguments, insufficient resources, and transmission errors. The errors that IP must report to ULPs are to be determined for each implementation.

6. UPPER LAYER SERVICE/INTERFACE SPECIFICATION

6.1 Description. This section specifies the IP services provided to upper layer protocols and the interface through which these services are accessed. The first part defines the interaction primitives and interface parameters for the upper interface. The second part contains the abstract machine specification of the upper layer services and interaction discipline.

6.2 Interaction primitives. An interaction primitive defines the purpose and content of information exchanged between two protocol layers. Primitives are grouped into two classes based on the direction of information flow. Information passed downward, in this case from a ULP to IP, is called a service request primitive. Information passed upward, in this case from IP to a ULP, is called a service response primitive. Interaction primitives need not occur in pairs. That is, a service request does not necessarily elicit a service “response;” a service “response” may occur independently of a service request. The information associated with an interaction primitive falls into two categories: parameters and data. The parameters describe the data and indicate how the data are to be treated. The data are not examined or modified. The format of the parameters and data are implementation dependent and therefore not specified. A given IP implementation may have slightly different interaction primitives imposed by the execution environment or system design factors. In those cases, the primitives can be modified to include more information or additional primitives can be defined to satisfy system requirements. However, all IPs must provide at least the interaction primitives specified below to guarantee that all IP implementations can support the same protocol hierarchy.

6.2.1 Service request primitives. A single service request primitive supports IP's datagram service, the SEND primitive.

6.2.1.1 SEND. The SEND primitive contains complete control information for each unit of data to be delivered. IP accepts in a SEND at least the following information:

- source address - internet address of ULP sending data
- destination address - internet address of ULP to receive data
- protocol - name of the recipient ULP
- type of service indicators - relative transmission quality associated with unit of data
    - precedence - one of eight levels : (PO, Pl, P2, P3, P4, P5, P6, P7) where P0 <= Pl <= P2 <= P3 <= P4 <= P5 <= P6 <= P7
    - reliability - one of two levels : (R0, R1) where R0 <= R1

- delay - one of two levels : (D0, D1) where D0 <= D1

- throughput - one of two levels : (T0, T1) where T0 <= Tl

- identifier - value optionally provided by this ULP distin guishing this portion of data from others sent by this ULP.

- don't fragment indicator - flag showing whether LP can fragment data to accomplish delivery

- time to live - The value in seconds which indicates the maximum lifetime of data within the Internet. Time to live is decremented by one second for each gateway transversed.

- data length - length of data being transmitted

- option data - options requested by a ULP from following list: security, loose or strict source routing, record routing, stream identification, or timestamp (section 9.3.14).

- data - present when data length is greater than zero.

6.2.2 Service response primitives. A single service response primitive supports IP's datagram service, the DELIVER primitive.

6.2.2.1 DELIVER. The DELIVER primitive contains the data passed by a source ULP in a SEND, along with addressing, quality of service, and option information. IP passes In a DELIVER at least the following information:

- source address - Internet address of sending ULP

- destination address - Internet address of the recipient ULP

- protocol - name of recipient ULP as supplied by the sending ULP

- type of service indicators - relative transmission quality associated with unit of data

  - precedence - one of eight levels : (P0, P1, P2, P3, P4, P5, P6, P7) where P0 <= P1 <= P2 <= P3 <= P4 <= P5 <= P6 <= P7

  - delay - one of two levels : (D0, D1) where D0 <= D1

  - reliability - one of two levels : (R0, R1) where R0 <= R1

  - throughput - one of two levels : (T0, T1) where T0 <= T1

- data length - length of received data (possibly zero)

- option data - options requested by source ULP from following list: security, loose source routing, strict source routing, record routing, stream identification, or timestamps (section 6.2.14).

- data - present when data length is greater than zero.

In addition, a DELIVER must contain error report from IP either together with parameters and data listed above, or independent of that information.

6.3 Extended state machine specification of services provided to upper layer. The extended state machine defines the behavior of the entire service machine from the perspective of the upper layer protocol. An extended state machine definition is composed of a machine instantiation identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

6.3.1 Machine instantiation identifier. Each upper Interface state machine is uniquely identified by the four interaction primitive parameters: source address, destination address, protocol, and identifier. One state machine instance exists for the SEND and DELIVER primitives whose four parameters carry identical values.

6.3.2 State diagram. The upper interface state machine has a single state which never changes. No diagram is needed.

6.3.3 State vector. The upper interface state machine has a single state which never changes. No state vector is needed.

6.3.4 Data structures. For clarity in the events and actions section, data structures are declared for the interaction primitives and their parameters. A subset of Ada[1] data constructs, common to most high level languages, is used. However, a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

6.3.4.1 From ULP. The from ULP structure holds the interface parameters and data associated with the SEND primitive specified above. This structure directly corresponds to the from_ULP structure declared in 9.4.4.2 of the mechanism section. The from_ULP structure is declared as:

```
type from_ULP_type is
  record
    source_addr
    destination_addr
    protocol
    type_of_service is
```

[1]ada is a registered trademark of the Department of Defense (Ada Joint Program Office).

```
        record
            precedence
            delay
            throughput
            reliability
        end record;
      identifier
      dont_fragment
      time_to_live
      length
      options
      data
    end record;
```

6.3.4.2 To ULP. The to_ULP structure holds interface parameters and data associated with the DELIVER primitive as specified in section 6.2.2. This structure directly corresponds to the to ULP structure declared in 9.4.4.3 of the mechanism specification. The to_ULP structure is declared as:

```
type to_ULP_type is
  record
    source_addr
    destination_addr
    protocol
    type_of_service is
      record
          precedence
          delay
          reliability
          throughput

      end record;
    length
    options
    data
    error
  end record;
```

6.3.5 Event list. The events are drawn from the interaction primitives specified in section 6.2 above. An event is composed of a service primitive and an abstract timestamp to indicate the time of event initiation. The event list is as follows:

a. SEND(from_ULP) at time t

b. NULL - Although no service request is issued by a ULP, certain conditions within IP or lower layers produce a service response. These conditions can include duplication of data and subnet errors.

6.3.6 Events and Actions. The following section defines the set of possible actions elicited by each event.

6.3.6.1 EVENT = SEND (from ULP) at time t.

Actions:

1. DELIVER to_ULP at time t+N to the protocol designated by from_ULP.protocol at destination from_ULP.destination_addr with all of the following properties:

   a. The time elapsed during data transmission satisfies the time-to-live limit, i.e., N <= from_ULP.time_to_live.

   b. The quality of data transmission is at least equal to the relative levels specified by from_ULP.type_of_service.

   c. if (from_ULP.dont_fragment = TRUE) then IP fragmentation has not occurred in transit.

   d. if (from_ULP.options includes loose source routing) then to_ULP.data has visited in transit at least the gateways named by source route provided by SEND.

   e. if (from_ULP.options includes strict source routing) then to_ULP.data has visited in transit only the gateways named by source route provided by SEND.

   f. if (from_ULP.options includes record routing) then the list of nodes visited in transit is delivered in to_ULP.

   g. if (from_ULP.options includes security labeling) then the security label is delivered in to ULP.

   h. if (from_ULP.options includes stream identifier) then the stream identifier is delivered in to_ULP.

   i. if (from ULP.options includes internet timestamp) then the internet timestamp is delivered in to_ULP.

OR,

2. DELIVER to the protocol designated by from_ULP.protocol at source from_ULP.source_addr indicating one of the following error conditions:

   a. destination from_ULP.destination_addr unreachable

b. protocol from_ULP.protocol unreachable

c. if (from_ULP.dont_fragment = TRUE) then fragmentation needed but prohibited

d. if (from_ULP.options contains any option) then parameter problem with option.

OR,

3. no action

6.3.6.2 EVENT = NULL.

Actions:

1. DELIVER to the protocol designated by from_ULP.protocol at source from_ULP.source_addr indicating the following error condition:

a. error conditions in subnet layer

OR,

2. DELIVER to_ULP at time t+N to the protocol designated by from_ ULP.protocol at destination from_ULP.destination addr with all of the following properties:

a. The time elapsed during data transmission satisfies the time-to-live limit, i.e. N <= from_ULP.time_to_live.

b. The quality of data transmission is at least equal to the relative levels specified by from_ULP.type_of_ service.

c. if (from_ULP.dont_fragment = TRUE) then IP fragmentation has not occurred in transit.

d. if (from_ULP.options includes loose source routing) then to_ULP.data has visited in transit at least the gateways named by source route provided in SEND.

e. if (from_ULP.options includes strict source routing) then to_ULP.data has visited in transit only the gate-ways named by source route provided in SEND.

f. if (from_ULP.options includes record routing) then the list of nodes visited in transit in delivered in to_ULP.

g. if (from_ULP.options includes security labeling) then the security label is delivered in to_ULP.

h. if (from_ULP.options = includes stream identifier)
then the stream identifier is delivered in to_ULP.

i. if (from_ULP.options includes internet timestamp)
then the internet timestamp is delivered in to_ULP

7. SERVICES REQUIRED FROM LOWER LAYER

7.1 Description. This section describes the minimal services required of the subnetwork layer. The services required are: transparent data transfer between hosts within a subnetwork and error reporting. A description of each service follows.

7.2 Data transfer. The subnetwork layer must provide a transparent data transfer between hosts within a single subnetwork. Only the data to be delivered, and the necessary control and addressing Information should be required as input from IP. Intranet routing and subnetwork operation shall be handled by the subnetwork layer itself. The subnetwork need not be a reliable communications medium. Data should arrive with non-zero probability at a destination. Data say not necessarily arrive in the same order as it was supplied to the subnetwork layer, nor is data guaranteed to arrive error free.

7.3 Error reporting. The subnetwork layer shall provide reports to IP indicating errors from the subnetwork and lower layers as feasible. The specific error requirements of the subnetwork layer are dependent on the individual subnetworks.

8. LOWER LAYER SERVICE/INTERFACE SPECIFICATION

8.1 Description. This section specifies the minimal subnetwork protocol services required by IP and the interface through which those services are accessed. The first part defines the interaction primitives and their parameters for the lower interface. The second part contains the abstract machine specification of the lower layer services and interaction discipline.

8.2 Interaction primitives. An interaction primitive defines the purpose of information exchanged between two protocol layers. Two kinds of primitives, based on the direction of information flow, are defined. Service requests pass information downward; service responses pass information upward. These primitives need not occur in pairs, nor in a synchronous manner. That in, a request does not necessarily elicit a “response;” a “response” may occur independently of a request. The information associated with an interaction primitive falls into two categories: parameters and data. The parameters describe the data and indicate how the data are to be treated. The data are not examined or modified and the format of interaction primitive information is implementation dependent and is therefore not specified. A given IP implementation may have slightly different interfaces imposed by the nature of the subnetwork or execution environment. Under such circumstances, the primitives can be modified to either include more parameters or have additional primitives defined. However, all IPs must provide at least the interface specified below to guarantee that all IP implementations can support the same protocol hierarchy.

8.2.1 Service request primitives. A single service request primitive is required from the SNP, a SNP SEND primitive.

8.2.1.1 SEND. The SNP SEND contains an IP datagram, a destination, and parameters describing the desired transmission quality. The SNP receives in an SNP_SEND at least the following information:

- local destination address - local subnetwork address of destination host or gateway
- type of service indicators - relative transmission quality associated with the datagram
    - precedence - one of eight levels: (P0, P1, P2, P3, P4, P5, P6, P7) where P0 <= P1 <= P2 <= P3 <= P4 <= P5 <= P6 <=P7
    - reliability - one of two levels: (R0, R1) where R0 <= R1
    - delay - one of two levels: (D0, D1) where D0 <= D1
    - throughput - one of two levels: (T0, T1) where T0 <= T1
- length - size of the datagram
- datagram

8.2.2 Service response primitives. One service response primitive is required to support IP's datagram service, the SNP_DELIVER primitive.

8.2.2.1 SNP_DELIVER. The SNP_DELIVER contains only a datagram which is an independent entity containing an IP header and data. An IP receives in an SNP_DELIVER at least the following information:

- datagram

In addition, a SNP_DELIVER may contain error reports from the SNP, either together with a datagram or independent of one.

8.3 Extended state machine specification of services required from lower later. The extended state machine defines the behavior of the entire service machine with respect to the lower layer protocol. An extended state machine definition is composed of a machine instantiation Identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

8.3.1 Machine instantiation identifier. Each lower interface state machine is uniquely identified by the four values:

- source address
- destination address
- protocol
- identification

These values are drawn from header fields of the datagram passed by the SNP_SEND and SNP_DELIVER primitives. One state machine instance exists for the interaction primitives whose parameters carry the same values.

8.3.2 State diagrams. The lower interface state machine has a single state which never changes. No diagram is needed.

8.3.3 State vector. No state vector is needed for the lower interface state machine.

8.3.4 Data structures. For clarity in the events and actions section, data structures are declared for the interaction primitives end their parameters. These structures are declared in a subset of Ada composed of constructs common to most high level languages. However, a data structure say be partially typed or completely untyped where specific formats or data types are implementation dependent.

8.3.4.1 From SNP. The from SNP structure holds the interface parameters and datagram associated with the SNP_DELIVER primitive, as specified in section 8.2.2.1 This structure directly corresponds to the fromSNP structure declared in section 9.4.4.4 of the mechanism specification. The fromSNP structure is declared as:

```
type from_SNP_type is
  record
     source_destination_addr
     dtgm: datagram_type;
     error
  end record;
```

The dtgm element is itself a structure as specified below.

8.3.4.2 To SNP. The to_SNP structure holds the data and parameters associated With the SNP_SEND primitive specified in section 8.3.1. This structure directly corresponds to the to_SNP structure declared in section 9.4.4.5 of the mechanism specification. The to_SNP structure is declared as:

```
type to_SNP_type is
  record
    local_destination_addr
    type_of_service_indicators
    length
    dtgm: datagram_type;
  end record;
```

The dtgm element is itself a structure as specified below.

8.3.4.3 Dtgm. The dtgm structure holds a datagram made up of a header portion and a data portion as specified in section 9.3. A dtgm structure is declared as:

```
type datagram_type is
  record
    version:  HALF_OCTET;
    header length:  HALF_OCTET;
    type_of_service:  OCTET;
    total length:  TWO_OCTETS;
    identification:  TWO_OCTETS;
    dont_frag_flag:  BOOLEAN;
    more_frag_flag:  BOOLEAN;
    fragment_offset:  ONE_N_FIVE_EIGHTHS_OCTETS;
    time_to_live:  OCTET;
    protocol:  OCTET;
    header_checksum:  TWO_OCTETS;
    source_addr:  FOUR OCTETS;
    destination_addr:  FOUR OCTETS;
    options:  option_type;
    data:  array(l..DATA_LENGTH) of INTEGER;
  end record;

subtype HALF_OCTET is INTEGER range 0..15;
subtype OCTET is INTEGER range 0..255;
subtype ONE_N_FIVE_EIGHTHS_OCTETS is INTEGER range 0..8191;
subtype TWO_OCTETS is INTEGER range 0..65535;
subtype FOUR_OCTETS is INTEGER range 0..4294967295;
```

8.3.5 Event list. The events are drawn from the service primitives specified in section 5.1 above. An event is composed of a service primitive with its parameters and data.

a. SNP_SEND (to_SNP)

b. NULL - Although IP issues no service request, certain conditions within the subnet layer elicit a service response.

8.3.6 Events and actions. The following section defines the set of possible actions elicited by each event.

8.3.6.1 EVENT = SNP SEND (to SNP).

ACTIONS:

1. SNP_DELIVER Datagram to IP at local destination (LD) with all of the following properties:

   a. The quality of data transmission is at least equal to the relative levels specified by toSNP.type_of_service.

OR,

2. no action

8.3.6.2 EVENT = NULL.

ACTIONS:

1. SNP_DELIVER from_SNP indicating the following error condition:

   a. error conditions within the subnet layer

9. IP ENTITY SPECIFICATION

9.1 Description. This section defines the mechanisms of an IP entity supporting the services provided by the IP service machine. The first subsection motivates the specific mechanisms chosen and describes their operation. The second subsection defines the format and use of the IP header fields. The last subsection specifies an extended state machine representation of the protocol entity. The implementation of a protocol entity must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the entity mechanisms, there is always the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret.

9.2 Overview of IP mechanisms. The IP mechanisms are motivated by the IP services, described in section 5 are datagram delivery service, virtual network service, and error reporting service. Each service could be supported by any of a set of mechanisms. The selection of mechanisms is guided by design standards including simplicity, generality, flexibility, and efficiency. The following mechanism descriptions identify the service or services supported, discuss the design criteria used in selection, and explain how the mechanisms work.

9.2.1 Routing mechanism. IP contains an adaptive routing mechanism to support the delivery service. The routing mechanism uses the internet addressing scheme and internet topology data to direct datagrams along the best path between source and destination. The mechanism provides routing options for ULPs needing the flexibility to select routes and record routing information. A distinction is made between names. addresses, and routes. A name indicates the object sought, independent of physical location. An address indicates where the object is and a route indicates how to get there. It is the task of the upper layer protocols to map from names to addresses. The internet protocol maps from internet addresses to local subnet addresses to perform routing through the internet. It is the task of lower layer protocols to route the datagram to the appropriate local subnet destination addresses.

9.2.1.1 Internet addresses. Internet addresses have a fixed length of four octets (32 bits). An internet address begins with a network number followed by a local address (called the REST field). To provide for flexibility in assigning addresses to networks and allow for the large number of small to medium sized networks, there are four formats or classes of internet addresses. These classes are shown in the following diagram:

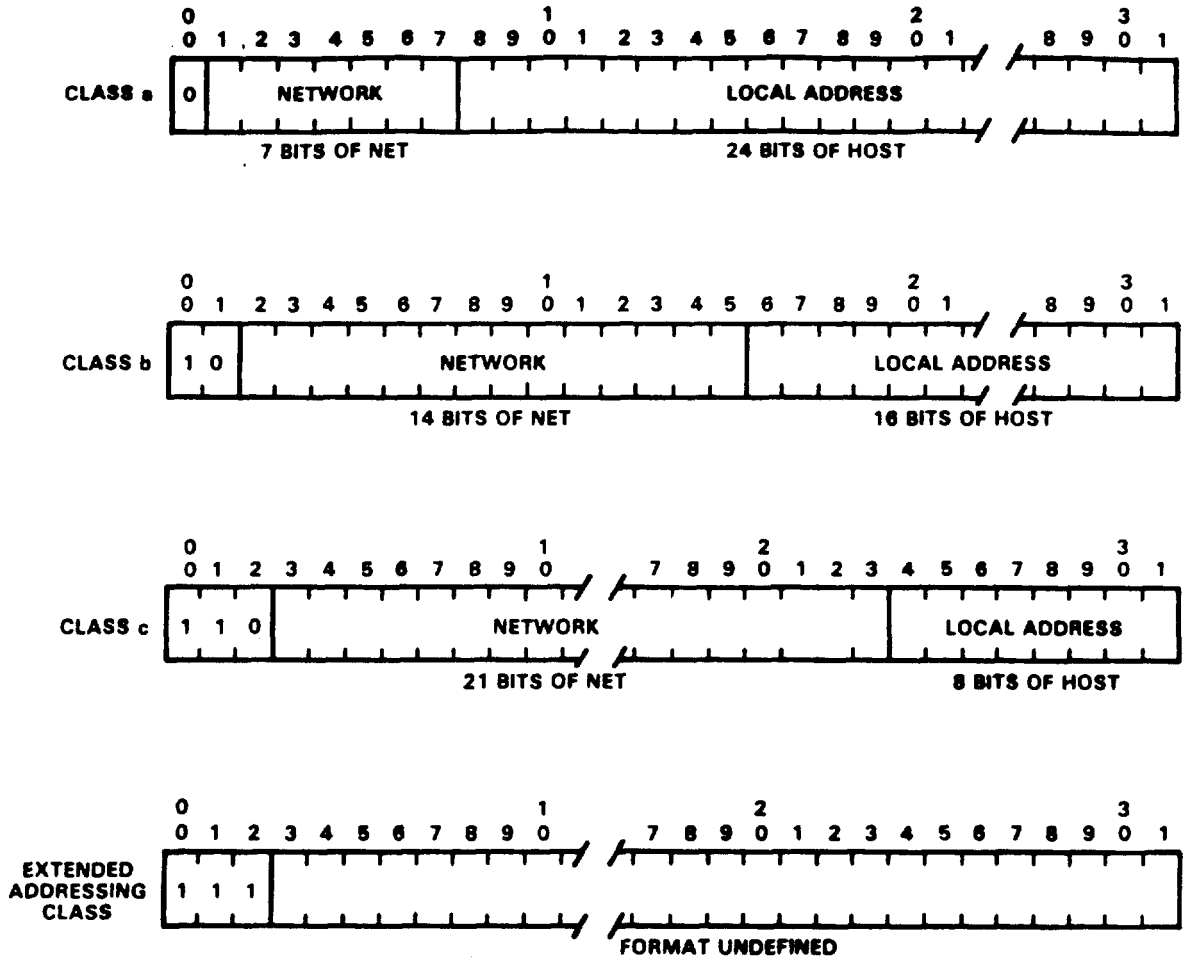


FIGURE 5. Internet addresses.

9.2.1.1.1 Internet addressing classes. In "class a," the high order bit is zero, the next seven bits specify the network, and the last 24 bits specify the local address. In 'class b.' the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address. In "class c," the high order three bits are one-one-zero, the next 21 bits are the network and the last eight bits are the local address. In the extended addressing class, the high order three bits are one-one-one, the next 29 bits have no defined format. The mapping between Internet addresses and local not addresses should allow a single physical host to act as several distinct Internet hosts. Also, some boats will have several physical interfaces (i.e., be multi-homed). That is, provision must be made for a host to have several physical interfaces to a subnetwork, with each having several logical Internet addresses.

9.2.1.1.2 Datascan routing. To route a datagram, an IP module examines the NETWORK field of the Internet address indicating the destination for the datagram. If the network number is the same as the IP module's subnetwork, the module uses the REST field of the Internet address to derive the local subnet address of the destination host. If the network number does not

match, the module determines a local subnet address of a gateway on the best path to the destination subnetwork. In turn, the gateway IP module derives the next local subnet address to either a host or gateway. In this way, the datagram is relayed through the internet to the destination host. In a static environment the routing algorithm is straightforward. However, internet topology tends to change due to hardware or software failure, host availability, or heavy traffic load conditions. Therefore, each host and gateway IP along the gateway route also uses its current knowledge of internet topology to make routing decisions.

9.2.1.2 Routing options. IP provides a mechanism, called source routing, to supplement the gateway's independent routing decisions. This mechanism allows an upper layer protocol to influence the gateway route in which a datagram traverses. The ULP can pass a list of internet addresses, called a source route list, as one of the SEND service request parameters. Each address on the list, except for the last, is an intermediate gateway destination. The last address on the list is the final destination. The source IP module uses its normal routing mechanism to transmit the datagram to the first address in the source route list. Then the gateway IP replaces source route list entry with its own address as known in the environment into which it is forwarding the datagram. Thus, the datagram follows the source route while recording its "inverse" or recorded route.

9.2.1.2.1 Routing types. Two kinds of source routing are provided by IP: loose and strict. With loose source routing, the host and gateway IP modules along the route may use any number of other intermediate gateways to reach the addresses in the source list. With strict source routing, the datagram must travel directly (i.e. through only the directly connected subnetwork indicated by each address) to each address on the source list. When the source route cannot be followed, the source host IP is notified with an error message, For testing or diagnostic purposes, a ULP can acquire a datagram's record route (independent of the source route option) by using the record route mechanism. The sending ULP supplies an empty record route list and indicates that the gateway route is to be recorded in transit. Then, as each gateway IP module on the gateway route relays the datagram, it adds its address an known in the succeeding environment to the record route list. The destination ULP receives the original datagram along with the record route list which, if reversed, provides a source route to the sending ULP. If more gateways are traversed than can be recorded La the list, the additional gateway addresses are not recorded. Problems with the record route option discovered in transit are reported to the source host IP. When using a routing option, the source ULP must provide a large enough route list to accommodate all the routing information expected. The size of a routing option does not change due to adding addresses.

9.2.2 Fragmentation and reassembly. IP contains a fragmentation mechanism for breaking a largo datagram into smaller datagrams. This solution to the problems arising from the difference between variable subnetwork capacity provides greater flexibility than legislating a restrictive datagram size that is sufficiently small for any subnetwork on the internet. This mechanism can be overridden using the "don't fragment" option to prevent fragmentation. IP also contains a reassembly mechanism which reverses the fragmentation to

enable delivery of intact data portions. Normally, fragmentation is performed only by the IP modules in gateways. There is no need for fragmentation of datagrams within the IP modules of hosts since the amount of data supplied by a source ULP can be limited, thereby avoiding datagrams which are too big to be transmitted through the subnetwork to which the host is attached. When an IP module encounters a datagram that is too big to be transmitted through a subnetwork, it applies its fragmentation mechanism. First, the module divides the data portion of the datagram into two or more pieces. The data must be broken on 8-octet boundaries. For each piece, it then builds a datagram header containing the identification, addressing, and options information needed. Fragmentation data is adjusted in the new headers to correspond to the data's relative position within the original datagram. The result is a set of small datagrams, called fragments, each carrying a portion of the data from the original large datagram. Section 9.4.6.3.7 defines the fragmentation algorithm.

9.2.2.1 Fragment routing. Each fragment is handled independently until the destination IP module is reached. The fragments may follow different gateway routes as Internet topology and traffic conditions change. They are also subject to further fragmentation if 'smaller-packet' subnetworks are subsequently traversed. Every IP module must be able to forward a datagram of 68 octets without further fragmentation. This size allows for a header length of up to 60 octets and the minimum data length of 8 octets.

9.2.2.2 Fragment reassembly. To reassemble fragments into the original datagram, an IP module combines all those received having the same value for the identification, source address, destination address, security, and protocol. IP allocates reassembly resources when a “first-to-arrive” fragment is recognized. Based on the fragmentation data in the fragment's header, the fragment is placed in a reassembly area relative to its position in the original datagram. When all the fragments have been received, the IP module passes the data in its original form to the destination ULP. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts send datagrams larger than 576 octets only if they have assurance that the destination is prepared to accept the larger datagrams. The number 576 is selected to allow a reasonable amount of data to be transmitted in addition to the required header Information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximum Internet header size is 60 octets, and a typical Internet header is 20 octets, allowing a margin for headers of upper layer protocols.

9.2.2.3 Fragment loss. Because the subnetwork may be unreliable, some fragments making up a complete datagram can be lost. IP uses the “time-to-live” data (explained in section 9.2.4 below) to set a timer on the reassembly process. If the timer expires before all the fragments have been collected, IP discards the partially reassembled datagram. Only the destination IP module should perform reassembly. This recommendation is intended to reduce gateway overhead and minimize the chance of deadlock. However, reassembly by private agreement between gateways is transparent to the rest of the

internet and in allowed. A ULP can prevent its data from being broken into smaller pieces during transmission. IF provides an override mechanism to prohibit fragmentation called “don’t fragment.” One example of the “don’t fragment” mechanism is the down line loading of a small host containing only a simple boot strap program to accept data from a datagram, storing it in memory, and executing it. Any internet datagram marked “don't fragment” cannot be fragmented by an IP module along the gateway route under any circumstances. If an IP module cannot deliver such a datagram to its destination without fragmenting it, the module discards the datagram and returns an error to the source IP. Note that fragmentation, transmission, and reassembly at the subnetwork layer is transparent to IP and can be used at any time.

9.2.3 Checksum. IF assumes the subnetwork layer to be unreliable regardless of the actual subnetwork protocol present. Therefore, IF provides a checksum mechanism supporting the delivery service to protect the IP header from transmission errors. The data portion is not covered by the IF checksum. If IP enforced a data checksum and discarded datagrams with data checksum failures, it could not support applications that require high throughput and can tolerate a low error rate. An IP module recomputes the checksum each time the IP header is changed. Changes occur in transit during time-to-live reductions, option updates (both explained below), and fragmentation. The checksum is currently a simple one's complement algorithm, and experimental evidence indicates its adequacy. However, the algorithm is provisional and may be replaced by a CRC procedure, depending on future experience.

9.2.4 Time-to-live. As mentioned in the routing discussion above, a datagram’s transmission path is subject to changes in internet topology and traffic conditions. Inadvertently, a datagram might be routed on a circuitous path to arrive at its destination after a considerable delay. Or, a datagram could loop through the same IP modules without making real progress towards its destination. Such “old datagrams” reduce internet bandwidth and waste processing time. To prevent these problems, IP provides a mechanism to limit the lifetime of a datagram, called time-to-live. Along with the other sending parameters, a ULP specifies a maximum datagram lifetime in second units. Each IF module on the gateway route decreases the time-to-live value carried in the IP header. If an IF module receives an expired datagram, it discards the datagram. The lifetime limit is in effect until the datagram’s data is delivered to the destination ULP. That is, if a datagram is fragmented during transmission, it can still expire during the reassembly process. Section 9.4.4.3 defines the reassembly algorithm use of the time-to-live data.

9.2.5 Type of service. In support of the virtual network service, the type of service mechanism allows upper layer protocols to select the transmission quality. IP passes the type of service (TOS) command set for service quality to the SNP where it is mapped into subnetwork-specific transmission parameters. Not every subnetwork supports all transmission services, but each SNP on the delivery path should make its best effort to match the available subnet services to the desired service quality. The TOS command set includes precedence level, a delay indication, a-throughput indication, and a reliability indication. Precedence is a measure of a

datagram's importance, A subnetwork say treat high precedence traffic more important than other traffic by preferentially allocating subnetwork resources especially during time of high load. The eight precedence levels begin with the lowest, Routine, and increase up to the two highest levels, Internetwork Control and Network Control. The highest precedence level, Network Control, is intended for use only within a subnetwork. The Internetwork Control level is intended for use by gateway control originators only. The actual use and access to these precedence levels is the responsibility of each subnetwork. Aside from precedence, the major service choice is a three-way tradeoff between low delay, high reliability, and high throughput. In many networks better performance for one of these parameters is coupled with worse performance for another. Except for very unusual cases, not more than two of these three indications should be set. The use of these service quality indications may increase the cost (in some sense) of the service. Section 9.3.15 specifies the legal values of the type of service indicators to be carried in the datagram header.

9.2.6 Data options. Motivated by the virtual network service, IP provides options to carry certain identification and timing data In a standard manner through the Internet. The use of this mechanism by the ULPs is optional, as the name implies, but all options must be supported by each IP implementation. The data options carry three kinds of information: security, stream identification, and timing. The security data is used by DoD hosts needing to transmit security information throughout the Internet in a standard manner. The security information (required if classified, restricted, or compartmented traffic is passed) includes security level, compartments, handling restrictions, and transmission control code. The stream identification option provides a way for a stream identifier to be carried both through stream-oriented subnetworks, for example SATNET, and subnets not supporting the stream concept.

9.2.6.1 Timing Information. Timing information, in the form of timestamps, is recorded by IP modules as the datagram traverses the internetwork to its destination. The source ULP provides a timestamp list and indicates timing information is to be recorded. The timestamp can be recorded In one of three formats. The first format requires each gateway IP module on the gateway route to register only its tinestamp in the next free list entry. The second format requires each gateway IP to register both Its Internet address and its tinestamp. The third format requires a timestamp to be registered only if the next list entry containing a prespecified Internet address matches the gateway IP's address. These formats are specified in section 9.2.15. A timestamp in a 32-bit value marking the current time in milliseconds since midnight Universal Time (UT). If the time is not available in milliseconds, or cannot be provided with respect to midnight UT, then any time may be Inserted If the high order bit of the timestamp field is set to one, indicating the use of a non-standard value. When using the timestamp option, the source ULP must provide a large enough list to accommodate all the timestamp information expected. The size of the option does not change due to adding timestamps. The initial contents of the timestamp list must be zero or Internet address/zero pairs. If the timestamp data area Is already full (the pointer exceeds the length) the datagram is forwarded without

inserting the timestamp, but the overflow count is incremented by one. If there is some room but not enough for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case, an ICMP parameter problem message may be sent to the source host. Errors encountered by the gateway IPs during timestamp processing are reported to the source IP.

9.2.7 Error report datagrams The error reporting service motivates a mechanism to generate and process error information. The error mechanism uses the datagram delivery service to transfer the error reports between IP modules.

9.3 Message format for poor exchanges. A summary of the contents of the IP header follows. Note that each tick mark represents a one bit position. Each field description below includes its name, an abbreviation, and the field size. Where applicable, the units, the legal range of values, and a default value appears.

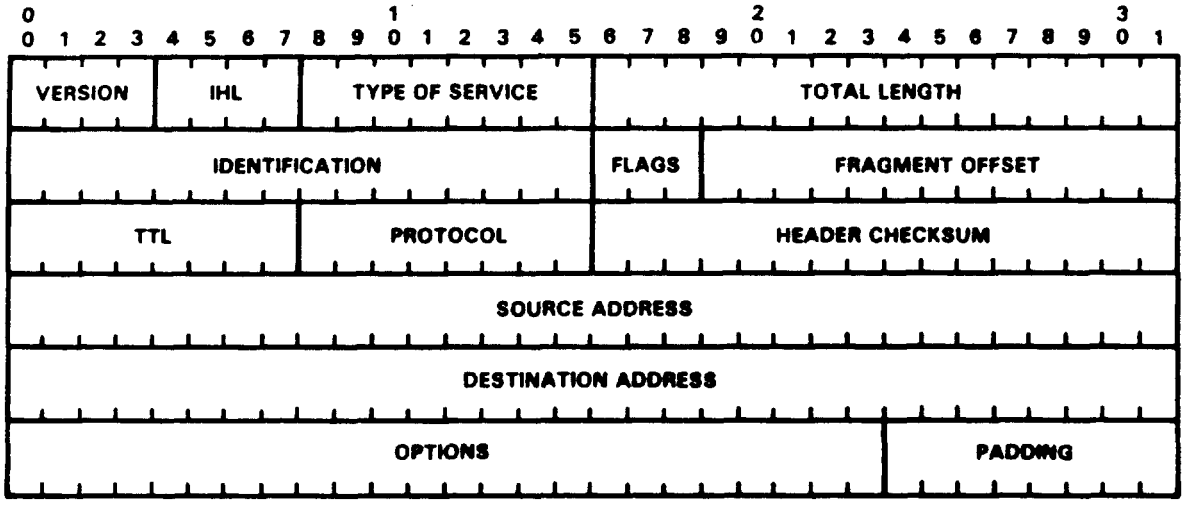


FIGURE 6. IP header format.

9.3.1 Version.

abbrev: VER field size: 4 bits

The Version field Indicates the format of the IP header. This document describes version 4.

9.3.2 Internet header length.

abbrev: IHL field size: 4 bits
units: 4-octet group range: 5 - 15 default: 5

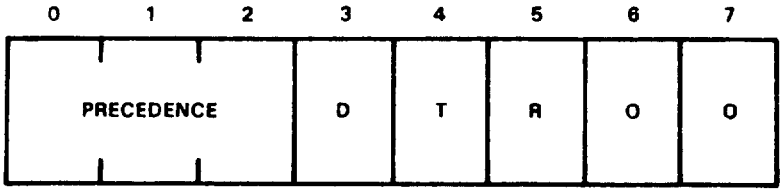
Internet Header Length is the length of the IP header in 32-bit words and points to the beginning of the data. Note that the minimum value for a correct header is 5.

9.3.3 Type of service.

abbrev: TOS field size: 8 bits

The Type of Service field contains the IP parameters describing the quality

of service desired for this datagram.



Bits 0-2: Precedence
Bit 3: Delay
Bits 4: Throughput
Bits 5: Reliability
Bit 6-7: Reserved for Future Use.

Precedence
111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

Delay
0 - normal
1 - low

Throughput
0 - normal
1 - high

Reliability
0 - normal
1 - high

FIGURE 7. Table of service field.

9.3.4 Total length.

abbrev: TL field size: 16 bits
units: octets range: 20 - 2**16-1 default: 20

Total Length is the length of the datagram, measured in octets, including header portion and the data portion of the datagram.

9.3.5 Identification.

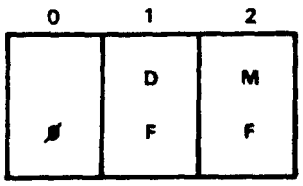
abbrev: ID field size: 16 bits

An identifying value used to associate fragments of a datagram. This value is usually supplied by the sending ULP as an interface parameter. If not, IP generates datagram identifications which are unique for each sending ULP.

9.3.6 Flags.

abbrev: none field size: 3 bits

This field contains the control flags “don’t fragment,” which prohibits IP fragmentation and, “more fragments,” which helps to identify a fragment's position in the original datagram.



Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

FIGURE 8. Control flags field.

9.3.7 Fragment offset.

abbrev: FO field size: 13 bits
units: 8-octet groups range: 0 - 8191 default: 0

This field indicates the positions of this fragment’s data relative to the beginning of the data carried in the original datagram. Both a complete datagram and a first fragment have this field set to zero. Section 9.2.2 describes the fragmentation mechanism.

9.3.8 Time-to-live.

abbrev: TTL field size: 8 bits
units: seconds range: 0 - 255(=4.25 mins) default: 15

This field indicates the maximum time the datagram is allowed to remain in the internet. If the value of this field drops to zero, the datagram should be destroyed. Section 9.2.4 describes the time-to-live mechanism.

9.3.9 Protocol.

abbrev: PROT field size: 8 bits

This field indicates which ULP is to receive the data portion of the datagram. The numbers assigned to common ULPs are available from the DoD Executive Agent for Protocols.

9.3.10 Header checksum.

abbrev: none field size: 16 bits

This field contains the checksum covering the IP header. The checksum mechanism is described in section 9.2.3.

9.3.11 Source address

abbrev: source field size: 32 bits

This field contains the internet address of the datagram's source host. Internet address formats are discussed in section 9.2.1.

9.3.12 Destination address.

abbrev: dest field size: 32 bits

This field contains the internet address of the datagram's destination host. Internet address formats are discussed in section 9.2.1.

9.3.13 Options.

abbrev: non e field size: variable

The option field is variable in length depending on the number and types of options associated with the datagram. The options mechanisms are discussed in sections 9.2.1 and 9.2.6. Options have two formats:

a. a single octet of option-type, or

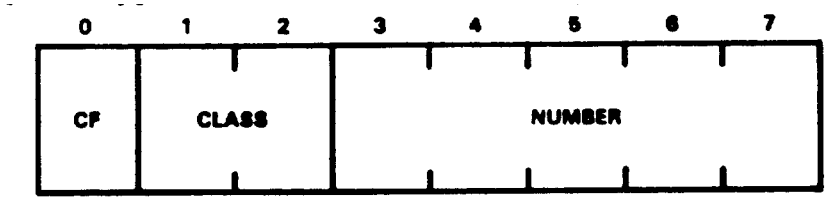
b. a variable length string containing:

1. an option-type octet,

2. an option-length octet - counting the option-type octet and option-length octet as well as the option-data octets, and

3. the actual option-data octets.

The option-type octet is viewed as having 3 fields:



bit 0 - copy flag
0 = not copied, 1 = copied
bits 1-2 - option class
000 = control
001 = reserved for future use
010 = debugging and measurement
011 = reserved for future use
bits 3-7 - option number (defined in the following table)

FIGURE 9. Fields in the option-type octet.

9.3.13.1 Internet options defined. The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0000	-	End of Option list: This option occupies only 1 octet; it has no length octet.
0	00001	-	No Operation: This option occupies only 1 octet; it has no length octet.
0	00010	11	Security: Used to carry security level, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DoD requirements.
0	00011	var.	Loose Source Routing: Used to route the datagram based on information supplied by the source.
0	01001	var.	Strict Source Routing: Used to route the datagram based on information supplied by the source.
0	00111	var.	Record Route: Used to trace the route a datagram takes.
0	01000	4	Stream ID: Used to carry the stream identifier.
2	00100	var.	Internet Timestamp: Used to accumulate timing information in transit.

9.3.14 Padding.

abbrev: none field size: variable (8 to 24 bits)

The IP header padding is used to ensure that the IP header ends on a 32-bit boundary. The padding field is set to zero.

9.3.15 Specific option definitions. Each option format is defined below. “Option type” indicates the value of the option-type octet, and “length” indicates the value of the length-octet if appropriate.

9.3.15.1 End of option list.

option type: 0 option length: N/A

This one-octet option marks the end of the option list when it does not coincide with the four-octet boundary indicated by the IP header length. This field is used following the last option, not the and of each option, and need only be used if the last option would not otherwise coincide with the end of the IP header. This option may be introduced or deleted upon fragmentation as needed.

9.3.15.2 No operation.

option type: 1 option length: N/A

This option may be used between options, for example, to align the beginning of a subsequent option on a 32-bit boundary. This option may be introduced or deleted upon fragmentation as needed.

9.3.15.3 Security.

option type: 130 option length: 11

This option (required if classified, restricted, or compartmented traffic is passed) provides a way for hosts to send Security level, Compartmentation, Handling Restriction Codes and User Groups (TCC) parameters through subnetworks in a standard manner. This option must be copied on fragmentation. This option appears at most once in a datagram.

The format for this option is as follows:

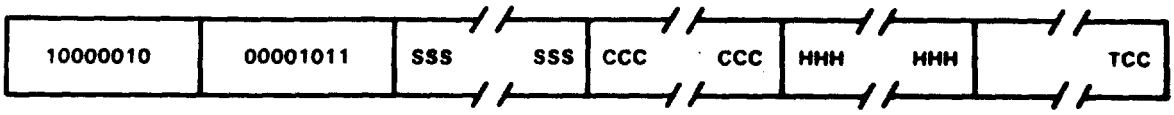


FIGURE 10. Security Option Format.

9.3.15.3.1 Security (S field).

length: 16 bits.

This field specifies one of 16 levels of security, eight of which are reserved for future use.

00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)

11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)

9.3.15.3.2 Compartments (C field).

length = 16 bits

This field contains an all zero value when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency (DIA).

9.3.15.3.3 Handling restrictions (H field).

length = 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, “Standard Security Markings.”

9.3.15.3.4 Transmission control code (TCC field).

length = 24 bits

This field provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs and are available from Headquarters, DCA (Code 530).

9.3.15.4 Loose source and record route.

option type: 131 option length: variable

The loose source route option provides a way for the source ULP of a datagram to supply routing information to be used by IP modules along the gateway route. At the same time, the “inverse” route to recorded in the option field. This option is not copied on fragmentation. It appears at most once in a datagram. The option begins with the option type code. The second octet is the option length which includes the option type octet, the length octet, the pointer octet, and the source route list. The third octet is a pointer into the route data indicating the octet which begins the next source address to be processed. The pointer in relative to this option, and its smallest legal value is 4. A loose source route list to composed of one or more internet addresses identifying intermediate gateways to be visited in transit. Each Internet address is 4 octets long. When a gateway in the source route list is visited, the gateway address (as known in the environment into which the datagram is being forwarded) replaces that list entry. The size of this option is fixed by the source. It cannot change to accommodate additional information. The routing options are described to section 9.2.1.1.

9.3.15.5 Strict source and record route.

option type: 137 option length: variable

The strict source route option provided a way for the source ULP of a datagram to name the exact set of IP modules to be visited along the gateway route. At the same time, the 'inverse' route is recorded in the option field. This option must be copied on fragmentation. It appears at most once in a datagram. The option begins with the option type code. The second octet is the option length which includes the option type octet, the length octet. the pointer octet, and the source route list. The third octet is a pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and its smallest legal value is 4. A strict source route list is composed of one or more internet addresses Identifying the gateways to be visited in transit. The datagram must visit exactly the gateways listed, traversing only the directly connected subnetworks indicated in the route list addresses. When a gateway in the source route list is visited, the gateway address (as known in the environment into which the datagram in being forwarded) replaces that list entry. The size of this option is fixed by the source. It cannot change to accommodate additional information. Routing options are described in section 9.2.1.1.

9.3.15.6 Record route.

option type: 7 option length: variable

The record route option provides a way to record a datagram's gateway route. This option is not copied on fragmentation. It appears at most once in a datagram. The option begins with the option type codes The second octet is the option length which includes the option type code. the length octet, and the return route list. The third octet is a pointer Into the route data indicating the octet which begins the next area to store a route address. The pointer in relative to this option, and the smallest legal value for the pointer is 4. A record route list is composed of a series of internet addresses. Each internet address is 4 octets long. The source ULP provides a route list with zero value entries. An each gateway is visited in transit, it registers Its address In the next free entry (indicated by the pointer). When the pointer is greater than the length. the record route list is full. No additional addresses are recorded, even if more are visited before arriving at the destinations The size of this option is fixed by the source. It cannot change to accommodate additional information. The routing options are described in section 9.2.1.1.

9.3.15.7 Stream Identifier.

option type: 136 option length: 4

This option provides a way for 16-bit stream identifiers to be carried through the internet for use by subnetworks supporting the stream concept such as the SATNET. The stream identifier appears in the third and fourth octets of

the option. This option must be copied on fragmentation. It appears at most once in a datagram.

9.3.15.8 Internet timestamp.

option type: 68 option length: variable

This option allows timing information to be gathered as a datagram travels through the internet to its destination. This option is not copied upon fragmentation and so appears only in the first fragment. This option may appear at most once in a datagram. The first octet is the option type. The second octet is the length of the option including the option type octet, the length octet, the pointer octet, the overflow/flag octet, and each timestamp or address/timestamp pair. The third octet is a pointer into the timestamp list identifying the octet beginning the space for the next timestamp. The pointer is relative to the beginning of this option; its smallest legal value is 5. The fourth octet is shared by overflow and format flag information. The first four bits record the number of IP modules that could not register timestamps due to lack of space. The second four bits indicate the format of the timestamp list:

0 - timestamps only, stored in consecutive 32-bit words
1 - each timestamp is preceded with the Internet address of the registering entity
2 - reserved for future use
3 - the internet address fields are prespecified by the source ULP. An IP module only registers its timestamp if its address matches the next one in the list.

The size of this option is fixed by the source. It cannot change to accommodate additional information. The internet timestamp option is described in section 9.2.6.

9.4 Extended state machine specification of IP entity. The IP entity is specified with an extended state machine made up of a set of states, a set of transitions between states, and a set of input events causing the state transitions. The following specification is made up of a machine instantiation identifier, a state diagram, a state vector, data structures, an event list, and a correspondence between events and actions. In addition, an extended state machine has an initial state whose values are assumed at state machine instantiation.

9.4.1 Machine instantiation identifier. Each datagram is an independent unit. Therefore, one state machine instance exists for each datagram. Each state machine is uniquely named by the four values, source address, destination address, protocol, and identification. These values are drawn from parameters of the interaction primitives specified in sections 6.2 and 8.2.

9.4.2 State diagram. The following diagram depicts a simplified IP state machine.

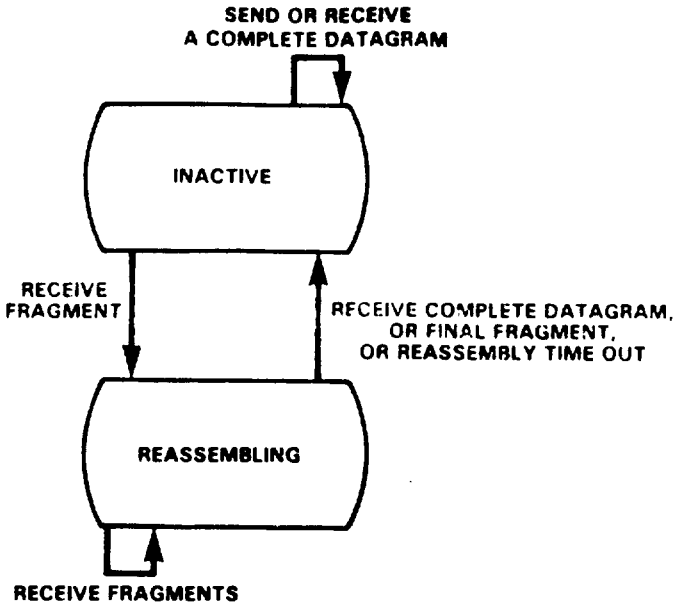


FIGURE 11. A simplified IP state machine.

9.4.3 State vector. A state vector consists of the following elements:

- STATE NAME = (inactive, reassembling)

- REASSEMBLY RESOURCES = control information and storage needed to reassemble fragments into the original datagram, including:

  a. reassembly map: a representation of each 8-octet unit of data and its relative location within the original datagram.

  b. timer: value of the reassembly-timer in unit seconds from 0 to 255.

  c. total data length: size of the data carried in datagram being reassembled.

  d. header: storage area for the header portion of the datagram being reassembled.

  e. data: storage area for the data portion of the datagram being reassembled.

A state machine's initial state is INACTIVE with unused reassembly resources.

9.4.4 Data structures. The IP state machine references certain data areas corresponding to the state vector, and each interaction primitive: SEND, DELIVER, SNP_SEND and SNP_DELIVER. For clarity in the events and actions section, data structures are declared in Ada for these data areas. However,

a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

9.4.4.1 State vector. The definition of an IP state vector appears in section 9.4.3 above. A state vector structure is declared as:

```
state_vector:  state_vector_type;

type state_vector_type is
  record
    state_name:  (INACTIVE, REASSEMBLING);
    reassembly_map
    timer
    total_data_length
    header
    data
  end record;
```

9.4.4.2 From ULP. The from_ULP structure holds the interface parameters and data associated with the SEND primitive, as specified in section 6.2.l. The from ULP structure is declared as:

```
from_ULP:  from_ULP_type;

type from_ULP_type  is
  record
    source_addr
    destination_addr
    protocol
    type_of_service is
      record
          precedence
          delay
          throughput
          reliability
      end record;
    identifier time_to_live
    dont_fragment
    length
    data
    options
  end record;
```

9.4.4.3 To ULP. The to_ULP structure holds Interface parameters and data associated with the DELIVER primitive, as specified to section 6.2.2. The to_ULP structure is declared as:

```
to_ULP : to_ULP_type;

type to_ULP_type_is
  record
    source_addr
```

```
            destination_addr
            protocol
            type_of_service is
              record
                  precedence
                  delay
                  throughput
                  reliability
              end record;
            length
            data
            options
            error
          end record;
```

9.4.4.4 From SNP. The from_SNP structure holds the interface parameters and datagram associated with the SNP_DELIVER primitive, as specified in section 8.3.2. The from_SNP structure is declared as:

```
type from_SNP_type is
  record
    local_destination_addr
    dtgm:  datagram_type;
    error
  end record;
```

The dtgm element is itself a structure as specified below.

9.4.4.5 To SNP. The to_SNP structure holds the data and parameters associated with the SNP_SEND primitive specified in section 8.3.1. The to_SNP structure is declared as:

```
type to_SNP_type is
  record
    local_destination_addr
    type_of_service_indicators
    length
    dtgm:  datagram_type;
  end record;
```

The dtgm element is itself a structure as specified below.

9.4.4.6 Dtgm. A dtgm structure holds a datagram made up of a header portion and a data portion as specified in section 9.3. A dtgm structure is declared as:

```
type datagram_type is
  record
    version:  HALF_OCTET;
    header_length:  HALF_OCTET;
    type_of_service:  OCTET;
    total_length:  TWO_OCTETS;
```

```
        identification:  TWO_OCTETS;
        dont_frag_flag:  BOOLEAN;
        more_frag_flag:  BOOLEAN;
        fragment_offset:  ONE_N_FIVE_EIGHTHS_OCTETS;
        time_to_live:  OCTET;
        protocol:  OCTET;
        header_checksum:  TWO_OCTETS;
        source_addr:  FOUR_OCTETS;
        destination_addr:  FOUR_OCTETS;
        options:  OPTION_TYPE;
        data:  array(l..DATA_LENGTH) of INTEGER;
      end record;

    subrecord HALF_OCTET is INTEGER range 0..15;
    subrecord OCTET is INTEGER range 0..255;
    subrecord ONE_N_FIVE_EIGHTHS_OCTETS is INTEGER range 0..8191;
    subrecord TWO_OCTETS is INTEGER range 0..65535;
    subrecord FOUR_OCTETS is INTEGER range 0..4294967296;
    subrecord OPTION_TYPE is zero or more of the following:
                             security;
                             loose source routing;
                             strict source routing;
                             record route;
                             stream identifier;
                             internet timestamp;
```

9.4.5 Event list. The event list is made up of the interaction primitives specified in sections 6.2 and 8.2 and the services provided by the execution environment defined In section 10. The following list defines the set of possible events in an IP state machine:

a. SEND from ULP -- A ULP passes interface parameters and data to IP for delivery across the Internet (see section 6.2.1.1).

b. SNP_DELIVER from SNP -- SNP passes to IP a datagram received from subnetwork protocol (see section 8.2.2.1).

c. TIMEOUT -- The timing mechanism provided by the execution environment Indicates a previously specified time interval has elapsed (see section 10.3).

9.4.6 Events and actions. This section is organized In three parts. The first part contains a decision table representation of state machine events and actions. The decision tables are organized by state; each table corresponds to one event. The second part specifies the decision functions appearing at the top of each column of a decision table. These functions examine attributes of the event and the state vector to return a set of decision results. The results become the elements of each column. The third part specifies action procedures appearing at the right of every row. Each row of the decision table combines the decision resultants determine appropriate event processing, These procedures specify event processing algorithms in detail.

9.4.6.1 Events and actions decision tables.

9.4.6.1.1 State = inactive, event is SEND from ULP.

TABLE I. Inactive state decision table when event is SEND from ULP.

Actions:

ULP PARAMS VALID?	WHERE DEST ?	NEED TO FRAG?	CAN FRAG ?	
NO	d	d	d	ERROR TO ULP (PARAM_PROBLEM)
YES	ULP	d	d	LOCAL DELIVERY
YES	REMOTE	NO	d	BUILD & SEND
YES	REMOTE	YES	NO	ERROR TO ULP (CAN'T_FRAG)
YES	REMOTE	YES	YES	FRAGMENT & SEND

Comments:

A ULP passes data to IP for Internet delivery. IP validates the Interface parameters, determines the destination, and dispatches the ULP data to its destination.

Legend

d = "don't care" condition

9.4.6.1.2 State = inactive, event is SNP DELIVER from SNP.

TABLE II. Inactive state decision table when event is SNP DELIVER from SNP.

Actions:

CHECK-SUM VALID?	SNP PARAMS VALID?	TTL VALID ?	WHERE TO ?	A FRAG ?	ICMP CHECK-SUM?	
NO	d	d	d	d	d	DISCARD
YES	NO	d	d	d	d	ERROR TO SOURCE (PARAM_PROBLEM)
YES	YES	NO	d	d	d	ERROR TO SOURCE (EXPIRED_TTL)
YES	YES	YES	ULP	NO	d	REMOTE DELIVERY
YES	YES	YES	ULP	YES	d	REASSEMBLE; STATE: = REASSEMBLING
YES	YES	YES	ICMP	NO	NO	DISCARD
YES	YES	YES	ICMP	NO	YES	ANALYZE
YES	YES	YES	ICMP	YES	d	REASSEMBLE; STATE: = REASSEMBLING
YES	YES	YES	REMOTE	d	d	ERROR TO SOURCE (HOST_UNREACH)

Comments:

The SNP has delivered a datagram from another IP. IP validates the datagram header, and either delivers the data from a complete datagram to its destination within the host or begins reassembly for a datagram fragment.

Legend
d = "don't care" condition

9.4.6.1.3 State = reassembling, event is SNP DELIVER from SNP.

TABLE III. Reassembling state decision table when event is SNP DELIVER from SNP.

Actions:

CHECK-SUM VALID?	SNP PARAMS VALID?	TTL VALID ?	WHERE TO ?	A FRAG ?	REASS DONE ?	ICMP CHECK-SUM?	
NO	d	d	d	d	d	d	DISCARD
YES	NO	d	d	d	d	d	ERROR_TO_SOURCE (PARAM_PROBLEM)
YES	YES	NO	d	d	d	d	ERROR_TO_SOURCE (EXPIRED_TTL)
YES	YES	YES	ULP	NO	d	d	REMOTE_DELIVERY; STATE: = INACTIVE
YES	YES	YES	ULP	YES	NO	d	REASSEMBLE
YES	YES	YES	ULP	YES	YES	d	REASS_DELIVERY; STATE: = INACTIVE
YES	YES	YES	ICMP	NO	d	NO	DISCARD
YES	YES	YES	ICMP	NO	d	YES	ANALYZE; STATE: = INACTIVE
YES	YES	YES	ICMP	YES	NO	d	REASSEMBLE
YES	YES	YES	REMOTE	d	d	d	ERROR_TO_SOURCE (HOST_UNREACH)

Comments:

The SNP has delivered a datagram associated with an earlier received datagram fragment. IP validates the header and either continues the reassembly process with the datagram fragment or delivers the data from the completed datagram to its destination within the host.

Legend
d = "don't care" condition

9.4.6.1.4 State = Inactive, event is TIMEOUT.

Actions: reassembly timeout; state: = INACTIVE

Comment:

The time-to-live period of the datagram being reassessed has elapsed. The incomplete datagram is discarded; the source IP is Informed.

9.4.6.2 Decision table functions. The following functions examine Information contained in interface parameters, interface data, and the state vector to make decisions. These decisions can be thought of as further refinements of the event and/or state. The return values of the functions represent decisions made. The decision functions appear in alphabetical order.

9.4.6.2.1 A frag? The a_frag function examines certain fields in an

incoming datagram’s header to determine whether the datagram is a fragment of a larger datagram. The data effects of this algorithm are:

a. Data examined only:

```
from_SNP.dtgm.fragment_offset
from_SNP.dtgm.more_frag_flag
```

b. Return values:

```
NO - the datagram has not been fragmented
YES - the datagram is a part of a larger datagram

if ((from_SNP.dtgm.fragment_offset = 0)     --contains the
                                              beginning
and (from_SNP.dtgm.more_frag_flag = 0)) --and the end
                                              of the data

then return NO   --therefore it Is an unfragmented datagram

else return YES; --otherwise it contains only a portion of
                   the data
                 --and is a fragment.
end if;
```

9.4.6.2.2 Can frag? The can_frag function examines the “don't fragment” flag of the interface parameters allowing fragmentation. The data effects of this function are:

a. Data examined only:

```
from_ULP.dont_fragment
```

b. Return values:

```
NO - “don’t fragment” flag is set, preventing fragmen-
     tation
YES - “don’t fragment” flag is NOT set, to allow frag-
     mentation

if (from_ULP.dont_fragment = TRUE)
then return NO
else return YES
end if;
```

9.4.6.2.3 Checksum valid? The checksum valid function examines an incoming datagram's header to determine whether it is free from transmission errors. The data effects of this function are:

a. Data examined only:

```
from_SNP.dtgm.version
from_SNP.dtgm.header_length
from_SNP.dtgm.type_of_service
from_SNP.dtgm.total_length
from_SNP.dtgm.identification
from_SNP.dtgm.dont_frag_flag
from_SNP.dtgm.more_frag_flag
from_SNP.dtgm.fragment_offset
from_SNP*dtgm.time_to_live
from_SNP.dtgm.protocol
from_SNP.dtgm.source_addr
from_SNP.dtgm.destination_addr
from_SNP.dtgm.options
```

b. Return values:

```
NO   -- checksum did not check, indicating header fields

        contain errors
YES  -- checksum was consistent

   --The checksum algorithm to the 16-bit one's complement
   --of the one's complement sum of all 16-bit words in
   --the IP header.  For purposes of computing the checksum,
   --the checksum field is set to zero.

       --implementation dependent action
```

9.4.6.2.4 Icmp checksum? The icmp_checksum function computes the checksum of the ICMP control message carried in the data portion of the incoming datagram. The data effects of this procedure are:

a. Data examined:

```
from_SNP.dtgm.data
```

b. Return values:

```
NO   -- checksum did not check indicating the control
        message contains errors
YES  -- checksum was consistent

   --The checksum algorithm in the 16-bit one's complement of
   --the one's complement sum of all 16-bit words
   --in ICMP control message.  For purposes of computing the
     checksum,
```

```
    --the checksum field (octets 2-3) is set to zero.

           --implementation dependent action
```

9.4.6.2.5 Need to frag? The need_to_frag function examines the interface parameters and data from a ULP to determine whether the data can be transmitted as a single datagram or must be transmitted as two or more datagram fragments. The data effects of this function are:

a. Data examined only:

```
           from_ULP.length
           from_ULP.options
```

b. Return values:

```
           NO - one datagram is small enough for the subnetwork
           YES - datagram fragments are needed to carry the data

    --Compute the datagram's length based on the length of data,
    --the length of options, and the standard datagram header size.

     if (( from_ULP.length + (number of bytes of option data)
             + 20) > maximum transmission unit of the local
             subnetwork)
     then return YES
     else return NO;
     end if;
```

9.4.6.2.6 Reass done? The reass_done function examines the incoming datagram and the reassembly resources to determine whether the final fragment has arrived to complete the datagram being reassembled. The data effects of this function are:

a. Data examined only:

```
           state_vector.reassembly_map
           state_vector.total_data_length
           from_SNP.dtgm.total_length
           from_SNP.dtgm.more_frag_flag
           from_SNP.dtgm.header_length
```

b. Return values:

```
           NO - more fragments are needed to complete reassembly
           YES - this is the only fragment needed to complete
                 reassembly

    --The total data length of the original datagram, as computed
    --from “tail” fragment, must be known before completion is
    --possible.
```

```
if (state_vector.total_data_length = 0)
then
  --Check incoming datagram for “tail.”

    if (from_SNP.dtgm.more_frag_flag = FALSE)
    then
      --Compute total data length and see if data in
      --this fragment fill out reassembly map.

        if (state_vector.reassembly_map from 0 to
             (((from_SNP.dtgm.total_length -         -- total data
                (from_SNP.dtgm.header_length*4)+7)/8)-- length
             +7)/8 is set)
        then return YES;
        end if;

    else
      --Reassembly cannot be complete if total data length unknown.
         return NO;
    end if;

 else --Total data length is already known.  See if data
      --in this fragment fill out reassembly map.

      if ( all reassembly map from 0 to
              (state_vector.total_data_length+7)/8 is set)
      then return YES;  --  final fragment
      else return NO;   --   more to come
      end if;
 end if;
```

9.4.6.2.7 SNP params valid? The SNP_params_valid function examines the interface parameters and the datagram received from the local subnetwork protocol to see if all values are within legal ranges and no errors have occurred. The data effects of this function are:

a. Data examined only:

```
from_SNP.dtgm.version
from_SNP.dtgm.header_length
from_SNP.dtgm.total_length
from_SNP.dtgm.protocol
other information/errors from SNP
```

b. Return values:

```
NO - some value or values are illegal or an error has
     occurred
YES - examined values are within legal ranges and no
      errors have occurred
```

```
        if ( --The current IP header version number is 4.
            (from_SNP.dtgm.version /= 4)

              --the minimal IP header is 5 32-bit units in length.
          or (from SNP.dtgm.header_length < 5)

              --The smallest legal datagram contains only a header and is
              --20 octets in length.
          or (from_SNP.dtgm.total_length < 20)

              --The legal protocol identifiers are
              --available from the DoD Executive Agent for Protocols.
          or (from_SNP.dtgm.protocol is not one of the acceptable identi-
             fiers)

          )

        then return NO

        else if (any implementation dependent values received from the
                 SNP are illegal or indicate error conditions)
            then return NO
            else return YES;   --Otherwise, all values look good.
            end if;
        end if;
```

9.4.6.2.8 TTL valid? The TTL_valid function examines the IP header time-to-live field of an incoming datagram to determine whether the datagram has exceeded its allowed lifetime. The data effects of this function are:

a. Data examined only:

from_SNP.dtgm.time-to-live

b. Return values:

NO - the datagram has expired
YES - the datagram has some life left in it

```
        --Decrement from_SNP.dtgm.time_to_live field by the maximum
        --of either the amount of time elapsed since the last IP module
        --handled this datagram (if known) or one second.

         if ( from_SNP.dtgm.time_to_live
                - maximum (number of seconds elapsed since last IP, 1)
             <= 0 )
         then return NO
         else return YES;
```

9.4.6.2.9 ULP params valid? The ULP_params_valid function examines the interface parameters received from a ULP to see if all values are within legal ranges and desired options are supported. The data effects of this function are:

a. Data examined only:

```
from_ULP.time_to_live
from_ULP.options
```

b. Return values:

```
NO - some value is illegal or a desired option is not
     supported.
YES - examined values are within legal ranges and desired
      options can be supported.
```

```
if (
     --The time-to-live value must be greater than zero to
     --allow IP to transmit it at least once.

       (from_ULP.time_to_live < 0)

or --The options requested are inconsistent.

     --Implementation dependent action

or --Other implementation dependent values are invalid.
     --implementation dependent action)

then return NO

else return YES;

end if;
```

9.4.6.2.10 Where dest? The where_dest function determines the destination of an outgoing datagram by examining the destination address supplied by the ULP. The data effects of this function are:

a. Data examined only:

```
from_ULP.destination_addr
```

b. Return values:

```
ULP - destination is an upper layer protocol at this
      location
REMOTE - destination to some remote location
```

```
--Examine the destination address field of the datagram
  header.
```

```
if (from_SNP.dtgm.destination_addr /= this site's address)
then return REMOTE
also return ULP;
end if;
```

9.4.6.2.11 Where to? The where_to function determines the destination of the incoming datagram by examining the address fields and options fields of the datagram header. The data effects of this function are:

a. Data examined only:

```
from_SNP.dtgm.destination_addr
from_SNP.dtgm.protocol
from_SNP.dtgm.options
```

b. Return values:

```
ULP - destination is an upper layer protocol at this
      location
ICMP - destination is this IP module because the
       datagram carries an ICMP control message
REM0TE - destination is some remote location
```

```
--The source route influences the datagram's gateway route.

  if ((from_SNP.dtgm.options contains the source routing
       option) and (all source route list addresses have
       not been visited))
  then return REMOTE;
  end if;

--Examine the destination address field of the datagram
  header.

  if (from_SNP.dtgm.destination_addr /= this site's address)
  then
      --It's destined for another site.
       return REMOTE
  else
      --It's destined for this site.
       if (from_SNP.dtgm.protocol = the ICMP protocol identi-
         fier)
       then return ICMP
       else return ULP;
       end if;
  end if;
```

9.4.6.3 Decision table action procedures. The following action procedures represent the set of actions an IP state Machine should perform in response to

a particular event and internal state. These procedures have been organized and designed for clarity and are intended as guidelines. Although implementors may in fact reorganize for better performance, the data effects of the resulting implementations must not differ from those specified below.

9.4.6.3.1 Analyze. The analyze procedure examines datagrams containing ICMP control messages from other IP modules. In general, error handling is implementation dependent. However, guidelines are provided to identify classes of errors and suggest appropriate actions. The data effects of this procedure are:

a. Data examined:

from SNP.dtgm.protocol
from_SNP.dtgm.data

b. Data modified:

implementation dependent

For simplicity, it is assumed that the data area can be accessed as a byte array.

```
--Examine the first octet in the data portion to identify the
--error type and subsequent format.

begin

case from_SNP.dtgm[l] of

    when 3 =>   --Destination Unreachable Message

   --The errors in the “unreachable” class
   --should be passed to the ULP indicating data delivery
   --to the destination is unlikely if not impossible.
   --The second octet identifies what level was unreachable.

    case from_SNP.dtgm[2] of

      when 0 =>  --net unreachable

      when 1 =>  --host unreachable

      when 2 =>  --protocol unreachable

      when 3 =>  --port unreachable

      when 4 =>  --fragmantation needed and don’t fragment set
```

```
         when 5 =>  --source route failed

      end case;

   when 11 =>  --Time Exceeded Message

      --The “time-out” errors are usually not passed
      --to the ULP but should be recorded for network
      --monitoring uses.

      case from_SNP.dtgm of

        when 0 =>  --Time to live exceeded in transit

        when 1 =>  --Fragment reassembly time exceeded

      end case;

   when 12 =>  --Parameter Problem Message

      --This error to generated by a gateway IP to indicate
      --a problem in the options field of a datagram header.
      --Octet 5 contains a pointer which identifies
      --the octet of the original header containing the error.

   when 4  =>  --Source Quench Message

      --This message indicates that a datagram has been
      --discarded for congestion control.  The ULP should
      --be informed so that traffic can be reduced.

   when 5  =>  --Redirect Message

      --This message should result in a routing table update
      --by the IP module.  Octets 5-8 contain the new value
      --for the routing table.  It is not passed to the ULP.

   when 8  =>  --Echo Datagram

   when 0  =>  --Echo Reply Datagram

   when 13  =>  --Timestamp Datagram

   when 14  =>  --Timestamp Reply Datagram

   when 15  =>  --Information Request Message

   when 16  =>  --Information Reply Message

end  case;
```

9.4.6.3.2 Build&send. The build&send procedure builds an outbound datagram in the to_SNP structure from the interface parameters and data in from_ULP and passes it to the SNP for transmission across the subnet. The data-effects of this procedure are:

a. Data examined:

from_ULP.source_addr	from_ULP.time_to_live
from_ULP.destination_addr	from_ULP.dont_fragment
from_ULP.protocol	from_ULP.options
from_ULP.type_of_service	from_ULP.length
from_ULP.identifier	from_ULP.data

b. Data modified:

to_SNP.dtgm	to_SNP.type_of_service_indicators
to_SNP.length	to_SNP.local_destinatian_addr

```
--Fill in each IP header field with information from fromULP or
--standard values.

    to_SNP.dtgm.version:  = 4;     --Current IP version is 4.
    to_SNP.dtgm.type_of_service.indicators:  = fromULP.type_of_service;
    to_SNP.dtgm.identification:  = from_ULP.identifier; --If ID is not given
                                                --by ULP, the IP must
                                                --supply its own.
    to_SNP.dtgm.dont_frag_flag:  = from ULP.dont_fragment;
    to_SNP.dtgm.more_frag_flag:  = false;
    to_SNP.dtgm.fragment offset:  = false;
    to_SNP.dtgm.time_to_live:  = from_ULP.time_to_live;
    to_SNP.dtgm.protocol:  = from_ULP.protocol;
    to_SNP.dtgm.source_addr:  = from_ULP.source_addr;
    to_SNP.dtgm.destination_addr:  = from_ULP.destination_addr;
    to_SNP.dtgm.options:  = from_ULP.options;
    to_SNP.dtgm.header_length:  =  5 + (number of bytes of option data)/4;
    to_SNP.dtgm.total_length:  = (to_SNP.dtgm.header_length)*4
                                               + (from_ULP.length);

--Call compute_checksum to compute and set the checksum.

    compute checksum;

--And, fill in the data portion of the datagram.

    to_SNP.dtgm.data[0..from_ULP.length -1]:  = from_ULP.data[0..
                                                  from_ULP.length-1];
--Set the type of service and length fields for the SNP.

    to_SNP.type_of_servlce_indicators:  = to_SNP.dtgm.type_of_servlce;
    to_SNP.length:  = to_SNP.dtgm.total length;
```

```
  --Call the route procedure to determine a local destination
  --from the internet destination address supplied by the ULP.

   route;

  --Request the execution environment to pass the contents of toSNP
  --to the local subnetwork protocol for transmission.

   TRANSFER to_SNP to the SNP.
```

NOTE: The format of the from_ULP elements is unspecified, allowing an implementor to assign data types for the interface parameters. If those data types differ from the IP header types, the assignment statements above become type conversions.

9.4.6.3.3 Compute checksum. The compute checksum procedure calculates a checksum value for a datagram header so that transmission errors can be detected by a destination IP. The data effects of this procedure are:

a. Data examined:

```
to_SNP.dtgm.version
to_SNP.dtgm.header_length
to_SNP.dtgm.type_of_service
to_SNP.dtgm.total_length
to_SNP.dtgm.identification
to_SNP.dtgm.dont_frag_flag
to_SNP.dtgm.more_frag_flag
to_SNP.dtgm.fragment_offset
to_SNP.dtgm.time_to_live
to_SNP.dtgm.protocol
to_SNP.dtgm.source_addr
to_SNP.dtgm.destination_addr
to_SNP.dtgm.options
```

b. Data modified:

```
   to_SNP.dtgm.header_checksum

  --checksum algorithm is the 16-bit one's complement of
  --the one's complement sum of all 16-bit words
  --in the IP header.  For purposes of computing the checksum,
  --the checksum field is set to zero.

       --implementation dependent action
```

9.4.6.3.4 Computing icmp checksum. The compute_icmp_checksum procedure computes the checksum of the ICMP control message carried in the data portion of an outgoing datagram. The data effects of this procedure are:

a. Data examined:

```
to_SNP.dtgm.data
```

b. Data modified:

```
to_SNP.dtgm.data[2-3]

--The checksum algorithm is the 16-bit one's complement of
--the one's complement sum of all 16-bit words
--in ICMP control message.  For purposes of computing the
  checksum,
--the checksum field (octets 2-3) is set to zero.

      --implementation dependent action
```

9.4.6.3.5 Error to source. The error_to_source procedure formats and returns an error report to the source of an erroneous or expired datagram. The data effects of this procedure are:

a. Parameters:

```
error_param : (PARAM_PROBLEM, EXPIRED_TTL,
               PROTOCOL_UNREACH);
```

b. Data examined:

```
from_SNP.dtgm
```

c. Data modified:

```
to_SNP.dtgm          to_SNP.local_destination_addr
to_SNP.length        to_SNP.type_of_service_indicators

--Format and transmit an error datagram to the source IP.

to_SNP.dtgm.version          :  = 4;      --standard IP version
to_SNP.dtgm.header_length    :  = 5;      --standard header size
to_SNP.dtgm.type_of_service  :  = 0;      --routine service quality
to_SNP.dtgm.identification   :  = select new value;
to_SNP.dtgm.more_frag_flag   :  = FALSE;
to_SNP.dtgm.dont_frag_flag   :  = FALSE;
to_SNP.dtgm.fragment_offset  :  = 0;
to_SNP.dtgm.time_to_live     :  = 60;     --or value large enough to
                                          --allow delivery
to_SNP.dtgm.protocol         :  = this number will be assigned by
                                  DoD Executive Agent for Protocols;
to_SNP.dtgm.source_addr      :  = from SNP.dtgm.destination_addr;
to_SNP.dtgm.destination_addr :  = from_SNP.dtgm.source_addr;
```

```
    --The data section carries the ICMP control message.
    --The first octet identifies the message type, the remaining
    --octets carry related information.

      case error_param of

         where PARAM_PROBLEM =>
           to_SNP.dtgm.data[0]:  = 12;    --ICMP type - Parameter Problem
           to_SNP.dtgm.data[l]:  = 0;     --Code = problem with option
           to_SNP.dtgm.data[4]:  = position of error octet;

         where EXPIRED_TTL =>
           to_SNP.dtgm.data[0]:  = 11;    --ICMP type = Time Exceeded
           to_SNP.dtgm.data[1]:  = 0;     --Code = TTL exceed in transit

         where PROTOCOL_UNREACH =>
           to_SNP.dtgm.data[0]:  = 3;     --ICMP type = Dest. Unreachable
           to_SNP.dtgm.data[l]:  = 2;     --Code = protocol unreachable

      end case;

    --The bad datagram's header plus the first 64 bytes of its
    --data section (a total of "N" octets) is copied in following
    --the ICMP information.

      to_SNP.dtgm.data[8..N+3]:  = from_SNP.dtgm.data[0..N-1];
      to_SNP.dtgm.total_length:  = from_SNP.header_length*4 + N + 8;
      compute_icmp_checksum;

    --Compute checksum, determine the route for the error datagram,
    --the type of service indicators, and the datagram size for the SNP.

      compute_checksum;
      to_SNP.type_of_service_indicators:  = 0;
      to_SNP.length := to_SNP.dtgm.total_length;
      route;

    --Request the execution environment to pass the contents of toSNP
    --to the local subnet protocol for transmission.

      TRANSFER to_SNP to the SNP.
```

9.4.6.3.6 Error to ULP. The error_to_ULP procedure returns an error report to a ULP which has passed Invalid parameters or has requested a service that cannot be provided. The data effects of this procedure are:

a. Parameters:

```
        error_param: (PARAM_PROBLEM, CAN'T_FRAGMENT,
                      NET_UNREACH, PROTOCOL_UNREACH,
                      PORT_UNREACH);
```

b. Data examined:

implementation dependent

c. Data modified:

to_ULP.error
implementation dependent parameters

```
--The format of error reports to a ULP is implementation
--dependent.  However, included in the report should be
--a value indicating the type of error, and some information
--to identify the associated data or datagram.

to_ULP.error := error_param;
--implementation dependent action
```

9.4.6.3.7 Fragment&send. The fragment&send procedure breaks data that is too big to be transmitted through the subnetwork as a single datagram into smaller pieces for transmission in several datagrams. Normally, hosts do not send datagrams too big to go through their own network. The data effects of the procedure are:

a. Data examined only:

```
from_ULP.source_addr                from_ULP.length
from_ULP.destination_addr           from_ULP.data
from_ULP.protocol                   from_ULP.options
from_ULP.identifier          from_ULP.time_to_live
from_ULP.dont_fragment
```

b. Data modified:

```
to_SNP.dtgm              to_SNP.type_of_service_indicators
to_SNP.length
```

c. Local variables:

```
number of fragments -- number of small datagrams created
                       from user data
data_per_fragment -- the number of octets in each small
                     datagram
number_frag_blocks -- the number of 8-octet blocks in each
                      small datagram
data_in_last_frag -- the number of octets in the last
                     datagram
j -- loop counter for each fragment generated

--Compute the fragmentation variables

--The amount at data per fragment equals the max datagram size
  less
```

```
--the length of the datagram header.
  data_per_fragment:  = maximum subnet transmission unit
            - (20 + number of bytes of option data);

  number_frag_blocks:  = data_per_fragment/8;

  number_of_fragments:  = (from_ULP.length + (data_per_frag-
                          ment-1)) / data_per_fragment;

  data_in_last_frag:  = from_ULP.length modulo data_per_frag-
                        ment;

--Create the first fragment and transmit it to the SNP.

       to_SNP.dtgm.version:  = 4;
       to_SNP.dtgm.header_length:  = 5 + (number bytes of option
                                     data/4);
       to_SNP.dtgm.total_length:  = to_SNP.dtgm.header_length
                                    + data_per_fragment;
       to_SNP.dtgm.identification:  = from_ULP.identifier;
       to_SNP.dtgm.dont_frag_flag:  = from_ULP.dont_fragment;
                                      --this will be false
       to_SNP.dtgm.more_frag_flag:  = TRUE;
       to_SNP.dtgm.fragment_offset:  = 0;

       to_SNP.dtgm.time_to_live:  = from_ULP.time_to_live;
       to_SNP.dtgm.protocol:  = from ULP.protocol;
       to_SNP.dtgm.source_addr:  = from_ULP.source_addr;
       to_SNP.dtgm.destination_addr:  = from_ULP.destination_addr;
       to_SNP.dtgm.options:  = from_ULP.options;
       to_SNP.dtgm.data[0..data_per_fragment-1]:  =
                                    from_ULP.data[0..data_per_
                                    fragment-1];

--Set the datagram's header checksum field.
compute_checksum;

--Call route to determine the subnetwork address of the
--destination.
  route;

--Also set the length and type of service indicators.
   to_SNP.length := to_SNP.dtgm.total_length;
   to_SNP.type_of_service_indicators := to_SNP.dtgm.type_
   of_service;

--Request the execution environment to pass the first fragment
--to the SNP.
   TRANSFER to_SNP to the local subnetwork protocol.

--Format and transmit successive fragments.

  for j in 1..number_of_fragments-1 loop
```

```
--The header fields remain the same as in the first
--fragment, EXCEPT for:

  if (“copy” flag present in any options) --most signi-
                                          --ficant bit
                                          --of option
                                          --octet

  then --put ONLY “copy” options into options fields and
       --adjust length fields accordingly.

       to_SNP.dtgm.options:  = (options with “copy” flag);
       to_SNP.dtgm.header_length:  =5 +
                                   (number of copy options
                                   octets/4);

  else --only standard datagram header present

       to_SNP.dtgm.header_length:  = 5;

  end if;

--Append data and set fragmentation fields.
  if (j /= number_of_fragments-1)

  then --middle fragment(s)

    to_SNP.dtgm.more_frag_flag:  = TRUE;
    to_SNP.dtgm.fragment_offset:  = j*number_frag_blocks;
    to_SNP.dtgm.total_length:  = to_SNP.dtgm.header_length
                                    + data_per_fragment;
    to_SNP.dtgm.data[0..data_per_fragment-1]:  =
                          from_ULP.data[j*data_per_frag-
                          ments.. (j*data_per_fragment
                           + data_per_fragment-l)];

  else --last fragment

    to_SNP.dtgm.more_frag_flag:  = FALSE;
    to_SNP.dtgm.fragment_offset:  = j*number_frag_blocks;
    to_SNP.dtgm.total_length:  = to_SNP.dtgm.header_length*4
                                     + data_in_last_frag;
    to SNP.dtgm.data[0..data_in_last_frag-1]:
                 from_ULP[j*data_per_fragment..
                   (j*data_per_fragment+ data_in__
                   _last_frag-1)];
  end  if;

--Call checksum to set the datagram’s header checksum field.
   checksum;
```

```
            --Call route to determine the subnetwork address of the
            --destination.
              route;

            --Also set the length and type of service indicators.
               to_SNP.length := to_SNP.dtgm.total_length;
               to_SNP.type_of_service_indicators := to_SNP.dtgm.type_of_
               service;

            --Request the execution environment to pass this fragment
            --to the SNP.
                   TRANSFER to SNP to the local subnetwork protocol.

         end loop;
```

A fragmentation algorithm may vary according to implementation concerns but every algorithm must meet the following requirements:

a. A datagram must not be fragmented if dtgm.dont_frag_flag is true.

b. The amount of data in each fragment (except the last) must be broken on 8-octet boundaries.

c. The first fragment must contain all options carried by the original datagram, except padding and no-op octets.

d. The security, source routing, and stream identification options (i.e. marked with “copy” flag, MSB in option octet) must be carried by all fragments, if present in the original datagram.

e. The first fragment must have to_SNP.dtgm.fragment_offset set to zeros

f. All fragments, except the last, must have to_SNP.dtgm.more_frag_flag set true.

g. The last fragment must have the to_SNP.dtgm.more_frag_flag set false.

9.4.6.3.8 Local delivery. The local delivery procedure moves the interface parameters and data in the from ULP structure to the to ULP structure and delivers it to an in-host ULP. The data effects of this procedure are:

a. Data examined:

from_ULP.destination_addr	from_ULP.length
from_ULP.source_addr	from_ULP.data
from_ULP.protocol	from_ULP.options
from_ULP.type_of_service	

b. Data modified:

```
to_ULP.source_addr                  to_ULP.length
to_ULP.destination_addr             to_ULP.data
to_ULP.protocol                     to_ULP.options
to_ULP.type_of_service
```

```
--Move the interface parameters and data from the input
--structure, from_ULP, directly to the output structure,
--to_ULP, for delivery to a local ULP.

      from_ULP.destination_addr:  = to_ULP.destination_addr;
      from_ULP.source_addr     :  = to_ULP.source_addr';-
      from_ULP.protocol        :  = to_ULP.protocol;
      from_ULP.type_of_service :  = to_ULP.type_of_service;
      from_ULP.length          :  = to_ULP.length;
      from_ULP.data            :  = to_ULP.data;
      from_ULP.options         :  = to_ULP.options;

--Request the execution environment to pass the contents of
--to_SNP to the local subnet protocol for transmission.

    TRANSFER to_ULP to to_ULP.protocol.
```

9.4.6.3.9 Reassembly. The reassembly procedure reconstructs an original datagram from datagram fragments. The data effects of this procedure are:

a. Data examined:

```
from_SNP.dtgm
```

b. Data modified:

```
state_vector.reassembly_map
state_vector.timer
state_vector.total_data_length
state_vector.header
state_vector.data
```

c. Local variables:

```
j -- loop counter

data_in_frag -- the number of octets of data in received
fragment
```

```
        data_in_frag:  = (from_SNP.dtgm.total_length-from_SNP.
        dtgm.header_length*4);

      --Put data in its relative position in the data area of the
        state vector.

        state_vector.data[from_SNP.dtgm.fragment_offset*8..
              from_SNP.dtgm.fragment_offset*8+data_in_frag]:  =
                           from_SNP.dtgm.data[0..data_in_frag-1];

      --Fill in the corresponding entries of the reassembly map
      --representing each 8-octet unit of received data.

        for j in (from_SNP.dtgm.fragment_offset)..
                 ((from_SNP.dtgm.fragment_offset + data_in_frag +
                 7)/8) loop

          state_vector.reassembly_map[j]:  = 1;
        end loop;

      --Compute the total datagram length from the "tail-end"
      --fragment.

        if (from_SNP.dtgm.more_frag_flag = FALSE)
        then state_vector.total_data length:  =
                               from_SNP.dtgm.fragment_offset*8 +
                               data_in_frag;
        end if;

      --Record the header of the "head-end" fragment.

        if (from_SNP.dtgm.fragment offset = 0)
        then state_vector.header := from_SNP.dtgm;
        end if;

      --Reset the reassembly timer if its current value is less
      --than the time-to-live field of the received datagram.

        state_vector.timer:  = maximum
                 (from_SNP.dtgm.time_to_live, state_vector.timer);
```

A reassembly algorithm may vary according to implementation concerns, but each one must meet these requirements:

a. Every destination IP module must have the capacity to receive a datagram 576 octets in length, either in one piece or in fragments to be reassembled.

b. The header of the fragment with from_SNP.dtgm.fragment_offset equal to zero (i.e. the “head-end” fragment) becomes the header of the reassembling datagram.

c. The total length of the reassembling datagram is calculated from the fragment with from_SNP.dtgm.more_frag_flag equal to zero (i.e., the “tail-end” fragment).

d. A reassembly timer is associated with each datagram being reassem bled. The current recommendation for the initial timer setting is 15 seconds. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium. That is, data rate multiplied by timer value equals reassembly capacity (e.g.10Kb/s X 15secs = 15OKb).

e. As each fragment arrives, the reassembly timer is reset to the maxi mum of state_vector.reassembly_resources.timer and from SNP.dtgm.time _to_live in the incoming fragment.

f. The first fragment of the datagram being reassembled must contain all options, except padding and no-op octets.

g. The source_addr, destination_addr, protocol, and identifier of the first fragment received must recorded. All subsequent fragments’ source_addr, destination_addr, protocol, and identifier will be compared against those recorded. Those fragments which do not match will be discarded.

h. As each fragment arrives, the security and precedence fields, if available, must be checked. If the security level of the fragment does not match the security level of the datagram or if the prece- dence level of the fragment does not match the precedence level of the datagram, the datagram being assembled is discarded. Also, an error datagram is returned to the source IP to report the “mismatched security/precedence” error.

i. If the reassembly timer expires, the datagram being reassembled is discarded. Also, an error datagram is returned to the source IP to report the “time exceeded during reassembly” error.

9.4.6.3.10 Reassembled delivery. The reassembled delivery procedure decomposes the datagram that has been reassembled in the state vector into interface parameters and data, then delivers them to a ULP. The data effects of this procedure are:

a. Data examined:

state_vector.header.destination_addr
state_vector.header.source_addr
state_vector.header.protocol
state_vector.header.type_of_service

```
                state_vector.header.header_length
                state_vector.header.total_length
                state_vector.header.options
                state_vector.data
```

b. Data modified:

```
                to_ULP.destination_addr       to_ULP.length
                to_ULP.source_addr            to_ULP.data
                to_ULP.protocol               to_ULP.options
                to_ULP.type_of_service


            to_ULP.destination_addr:  = state_vector.header.destina-
                                        tion_addr;
            to_ULP.source_addr     :  = state_vector.header.source_addr;
            to_ULP.protocol        :  = state_vector.header.protocol;
            to_ULP.type_of_service :  = state_vector.header.type_of_
                                        service;
            to_ULP.length          :  = state_vector.header.total_
                                        length;
                                        - state_vector.header.header
                                          _length*4;
            to_ULP.options         :  = state_vector.header.options;
            to_ULP.data            :  = state_vector.data;
```

9.4.6.3.11 Reassembly timeout. The reassemble timeout procedure generates an error datagram to the source IP informing it of the datagram's expiration during reassembly. The data effects of the procedure are:

a. Data examined:

```
                state_vector.header
                state_vector.data
```

b. Data modified:

```
                to_SNP.dtgm                    to_SNP.type_of_service
                indicators
                to_SNP.length            to_SNP.header_length

            --Format and transmit an error datagram to the source IP.

             to_SNP.dtgm.version        :  = 4; --standard IP version
             to_SNP.dtgm.header_length  :  = 5; --standard header size
             to_SNP.dtgm.type_of_service:  = 0; --routine service quality

             to_SNP.dtgm.identification :  = new value selected
             to_SNP.dtgm.more_frag_flag :  = FALSE;
             to_SNP.dtgm.dont_frag_flag :  = FALSE;
```

```
        to_SNP.dtgm.fragment_offset :  = 0;
        to_SNP.dtgm.time_to_live    :  = 60;
        to_SNP:dtgm.protocol        :  = this number will be as-
                                       signed by the DoD Executive
                                       Agent for Protocols;
        to_SNP.dtgm.source_addr     :  = state_vector.header.desti-
                                        nation addr;
        to_SNP.dtgm.destination_addr:  = state_vector.header.source
                                        _addr;

    --If the fragment received is the first fragment, then the
    --data section carries the ICMP error message, the header of the
    --timed-out datagram, and its first 64 bytes of data.  If frag-
    --ment zero is not available then no time exceeded need be sent
    --at all.

      to_SNP.dtgm.data[0]:  = 12; --ICMP type = Time Exceeded
      to_SNP.dtgm.data[l]:  = 1;  --Code = fragment reassembly
                                    timeout

    --Copy in the timed-out datagram's header plus the first
    --64 bytes of its data section (assumed to be of length "N").

      to_SNP.dtgm.data[8..N+3] := state_vector[0..N-1];
      to_SNP.dtgm.total_length := to_SNP.header_length*4 + N + 8;
      compute_icmp_checksum;

    --Compute datagram's header checksum, determine the route for
    --the datagram, the type of service indicators, and the
    --datagram size for the SNP.

      compute_checksum;
      to_SNP.type_of_service_indicators := 0;
      to_SNP.length := to_SNP.dtgm.total_length;
      route;

    --Request the execution environment to pass the contents of
    --to_SNP to the local subnet protocol for transmission.

      TRANSFER to SNP to the SNP.
```

9.4.6.3.12 Remote delivery. The remote_delivery procedure decomposes a datagram arriving from a remote IP into interface parameters and data and delivers them to the destination ULP. The data effects of this procedure are:

a. Data examined:

```
    from_SNP.dtgm.source_addr
    from_SNP.dtgm.destination_addr
```

```
                from_SNP.dtgm.protocol
                from_SNP.dtgm.type_of_service
                from_SNP.dtgm.total_length
                from_SNP.dtgm.header_length
                from_SNP.dtgm.data
                from_SNP.dtgm.options
```

b. Data modified:

```
                to_ULP.destination_addr       to_ULP.length
                to_ULP.source_addr            to_ULP.date
                to_ULP.protocol               to_ULP.options
                to_ULP.type_of_service

            to_ULP.destination_addr:  = from_SNP.dtgm.destination_addr;
            to_ULP.source_addr     :  = from_SNP.dtgm.source_addr;
            to_ULP.protocol        :  = from_SNP.dtgm.protocol;
            to_ULP.type_of_service :  = from_SNP.dtgm.type_of_service;
            to_ULP.length          :  = from_SNP.dtgm.total_length -
                                          from_SNP.dtgm.header_length*4;
            to_ULP.data            :  = from_SNP.dtgm.data;
            to_ULP.options         :  = from_SNP.dtgm.options;
```

NOTE: The format of the to_ULP elements is unspecified, allowing an imple mentor to assign data types for the interface parameters. If those data types differ from the IP header types, the assignment statements above become type conversions.

9.4.6.3.13 Route. The route procedure examines the destination address and options fields of an outbound datagram in to_SNP to determine a local destination address. The data effects of this procedure are:

a. Data examined:

```
                to_SNP.dtgm.destination_addr
                to_SNP.dtgm.options
```

b. Data modified:

```
                to_SNP.local_destination_addr
                to_SNP.dtgm.options
```

The procedure:

```
     if (to_SNP.dtgm.options includes timestamp)
     then
    if (the next timestamp field in to_SNP.dtgm.options.timestamp
           is available)
     then
```

```
     --The timestamp or address/timestamp pair is inserted in
     --the next field in to_SNP.dtgm.options.timestamp.
   end if;
 end if;

if (the network id field of destination matches the network id
       of the local subset protocol
then
     --Translate the REST field of destination into the subnetwork
     --address of the destination on this subnet.
     --implementation dependent action
else

   if (to_SNP.dtgm.options includes security)
   then
    --Find the appropriate gateway with security level equal to
    --the security level of to_SNP.dtgm.options.security.   If
    --none exists, send error message.
   end if;

   if (to_SNP.dtgm.options includes loose source and record routing)
   then
   if (the network id field of next gateway in to_SNP.dtgm.option.
          loose_source matches the network of the local subnet
          protocol)
   then
        --The gateway address (as known in the environment into
        --which the datagram is being forwarded) replaces the
        --the network id field of next gateway in to_SNP.dtgm
        --.options.loose_source
   end if;
   end if;

   if (to_SNP.dtgm.options includes strict source and record routing)
   then

   if (the network id field of next gateway in to_SNP.dtgm.option.
          strict_source matches the network of the local subnet
          protocol)
   then
        --The gateway address (as known in the environment into
        --which the datagram is being forwarded) replaces the
        --network id field of next gateway in to_SNP.dtgm
        --.options.strict_source.
   else
        --The datagram cannot be forwarded and error message sent.
   end if;
   end if;
```

```
  if (to_SNP.dtgm.options includes record routing)
  then
     if (the next record route field in to_SNP.dtgm.options.record_
             _routing is available)
     then
          --The gateway address (as known in the environment into
          --which the datagram is being forwarded) replaces the
          --next record route field in to_SNP.dtgm.options.record
          --_routing.
     end if;
   end if;
   end if;

--Set the local destination interface parameter.

   to_SNP.local_destination_addr := (subnetwork address found above)
```

10. EXECUTION ENVIRONMENT REQUIREMENTS

10.1 Description. This section describes the facilities required of an execution environment for proper implementation and operation of the internet Protocol. Throughout this document, the environmental model portrays each protocol as an independent process. Within this model, the execution environment must provide two facilities: interprocess communication and timing.

10.2 Interprocess communication. The execution environment must provide an interprocess communication facility to enable independent processes to exchange variable-length units of information, called messages. For IP's purposes, the IPC facility is not required to preserve the order of messages. IP uses the IPC facility to exchange interface parameters and data with upper layer protocols across its upper interface and the subnetwork protocol across the lower interface. Sections 6 and 8 specify these interfaces.

10.3 Timing. The execution environment must provide a timing facility that maintains a real-time clock with units no coarser than 1 millisecond. A process must be able to set a timer for a specific time period and be informed by the execution environment when the time period has elapsed. A process must also be able to cancel a previously set timer. Two IP mechanisms use the timing facility. The internet timestamp carries timing data in millisecond units. The reassembly mechanism uses timers to limit the lifetime of a datagram being reassembled. In the mechanism specification this facility is called TIMEOUT.

Custodians:
Army - CR
Navy - OH
Air Force - 90

Preparing Activity:
DCA - DC

(Project IPSC-0167-01)

Review Activities:
Army - SC, CR, AD
Navy - AS, YD, MC, ON, ND, NC, EC, SA
Air Force - 1, 11, 13, 17, 99, 90

Other Interest:
NSA - NS
TRI-TAC-TT

APPENDIX A - DATA TRANSMISSION ORDER

The order of transmission of the header and data described in this document is resolved to the octet level, Whenever a diagram shown a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they ore numbered.

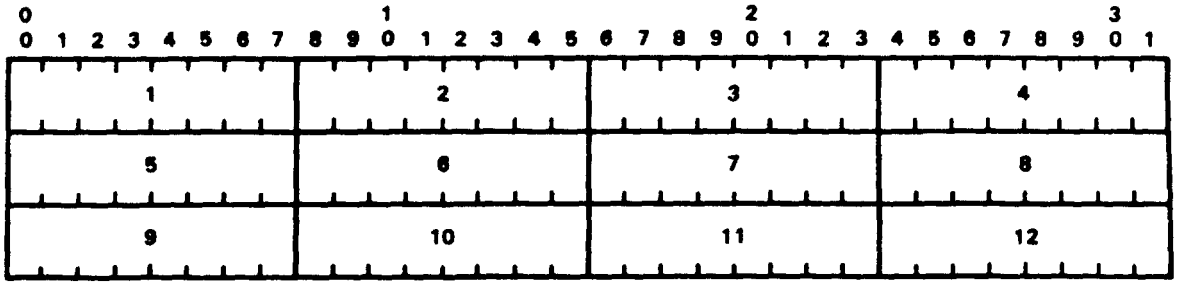


FIGURE 12. Transmission order of octets.

Whenever an octet represents a numeric quantity, the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).

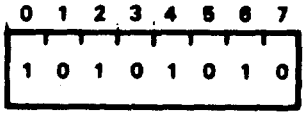


FIGURE 13. Significance of bits.

Similarly, whenever a multi-octet field represents a numeric quantity, the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

INSTRUCTIONS: In a continuing effort to make our standardization documents better, the DoD provides this form for use in submitting comments and suggestions for improvements. All users of military standardization documents are invited to provide suggestions. This form may be detached, folded along the lines indicated, taped along the loose edge *(DO NOT STAPLE)*, and mailed. In block 5, be as specific as possible about particular problem areas such as wording which required interpretation, was too rigid, restrictive, loose, ambiguous, or was incompatible, and give proposed wording changes which would alleviate the problems. Enter in block 6 any remarks not related to a specific paragraph of the document. If block 7 is filled out, an acknowledgement will be mailed to you within 30 days to let you know that your comments were received and are being considered.

NOTE: This form may not be used to request copies of documents, nor to request waivers, deviations, or clarification of specification requirements on current contracts. Comments submitted on this form do not constitute or imply authorization to waive any portion of the referenced document*(s)* or to amend contractual requirements.

(Fold along this line)

(Fold along this line)

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE $300

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 4966 Alexandria, VA

POSTAGE WILL BE PAID BY

Director
Defense Communications Agency
ATTN: Code J110
1860 Wiehle Avenue
Reston, VA 22090

STANDARDIZATION DOCUMENT IMPROVEMENT PROPOSAL

(See Instructions – Reverse Side)

1. DOCUMENT NUMBER	2. DOCUMENT TITLE
MIL STD 1777	

3a. NAME OF SUBMITTING ORGANIZATION	4. TYPE OF ORGANIZATION *(Mark one)*
	☐ VENDOR
b. ADDRESS *(Street, City, State, ZIP Code)*	☐ USER
	☐ MANUFACTURER
	☐ OTHER *(Specify)*: _____

5. PROBLEM AREAS

a. Paragraph Number and Wording:

b. Recommended Wording:

c. Reason/Rationale for Recommendation:

6. REMARKS

7a. NAME OF SUBMITTER *(Last, First, MI)* – Optional	b. WORK TELEPHONE NUMBER *(Include Area Code)* – Optional
c. MAILING ADDRESS *(Street, City, State, ZIP Code)* – Optional	8. DATE OF SUBMISSION *(YYMMDD)*

DD FORM 82 MAR 1426 PREVIOUS EDITION IS OBSOLETE.